(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 103259793 B
(45)授权公告日 2015.10.21

(21)申请号 201310159057.1

(22)申请日 2013.05.02

(73)专利权人 东北大学
地址 110819 辽宁省沈阳市和平区文化路3号巷11号东北大学

(72)发明人 敬茂华

(74)专利代理机构 北京联创佳为专利事务所（普通合伙） 11362
代理人 郭防

(51)Int.Cl.
H04L 29/06(2006.01)
H04L 12/26(2006.01)

(56)对比文件
CN 101645069 A,2010.02.10,
CN 101707513 A,2010.05.12,
CN 102523219 A,2012.06.27,
US 7689530 B1,2010.03.30,
Hiroki Nakahara等.A regular expression matching circuit: Decomposed non-deterministic realization with prefix sharing and multi-character transition.《Microprocessors and Microsystems》.2012,第36卷（第8期）,
殷珍珍．基于正则表达式的多模式匹配算法研究．《中国优秀硕士学位论文全文数据库 信息科技辑》.2012,

审查员 周萍

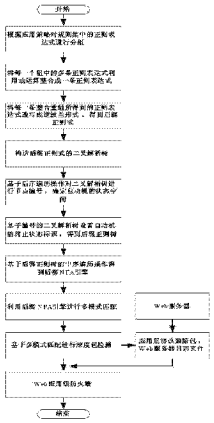
权利要求书2页 说明书10页 附图3页

(54)发明名称

基于后缀自动机正则引擎构造的深度包检测方法

(57)摘要

本发明公开了一种基于后缀自动机正则引擎构造的深度包检测方法，包括以下步骤：S1，入侵检测系统抽取攻击特征构建正则表达式；S2，构造后缀NFA引擎并利用其进行多模式匹配；S3，从Web服务器获取应用层协议数据包和Web服务器日志文件；S4，对上述协议数据包和日志文件进行深度包检测并将检测结果发送给防火墙。本发明用NFA方式实现了DFA的用单一自动机完成对多条正则表达式的匹配，解决了NFA不能实现多正则表达式匹配和DFA在实现多正则表达式匹配时的空间爆炸问题；有效地缩减了NFA的空间规模，解决了传统的NFA引擎构造方法所存在的空间浪费以及在执行模式匹配过程中的无效遍历问题，有效地缩短了深度包检测的响应时间，提高了系统的整体性能和效率。



CN 103259793 B

1.一种基于后缀自动机正则引擎构造的深度包检测方法，其特征在于，包括以下步骤：

S1，入侵检测系统抽取攻击特征构建正则表达式；

S2，构造后缀 NFA 引擎并利用其进行多模式匹配；

S3，从 Web 服务器获取应用层协议数据包和 Web 服务器日志文件；

S4，对上述协议数据包和日志文件进行深度包检测并将检测结果发送给防火墙；

S5，进行 IP 溯源，追踪到攻击源后，将攻击源的 IP 地址发送到防火墙进行包过滤。

2.根据权利要求 1 所述的基于后缀自动机正则引擎构造的深度包检测方法，其特征在于，步骤 S2 中所述的构造后缀 NFA 引擎的具体方法包括：

a. 对正则表达式进行分组；

b. 将每一组中的多条正则表达式利用或运算整合成一条正则表达式；

c. 将整合后的正则表达式改写成逆波兰形式，得后缀正则式；

d. 构造后缀正则式的二叉解析树；

e. 基于后序遍历操作对二叉解析树进行节点编号，确定自动机的状态空间；

f. 基于编号的二叉解析树设置自动机的终止状态标识，得后缀正则树；

g. 基于后缀正则树的中序遍历操作得后缀 NFA 引擎。

3.根据权利要求 2 所述的基于后缀自动机正则引擎构造的深度包检测方法，其特征在于，步骤 c 中所述的将整合后的正则表达式改写成逆波兰形式，得后缀正则式的具体方法包括：

正则表达式中按照原有顺序依次出现的输入符号作为后缀正则式从左至右输入的符号；

运算符号按其实际计算顺序紧跟在其运算对象后面出现；

将正则表达式中的小括号删除。

4.根据权利要求 2 所述的基于后缀自动机正则引擎构造的深度包检测方法，其特征在于，步骤 d 中所述的构造后缀正则式的二叉解析树的方法是：

1）后缀正则式中的所有输入符号作为二叉解析树的叶子节点，从左至右依次排列；

2）根据后缀正则式中运算符的逻辑顺序依次由叶子节点到根节点自底向上建立整棵二叉解析树，其中，或运算和连接运算为双目运算，其运算对象：输入符号所对应的叶子节点或某个运算的运算结果所对应的子树的根节点分别为其左右孩子；闭包运算为单目运算，其运算对象为其左孩子，右孩子空缺。

5.根据权利要求 2 所述的基于后缀自动机正则引擎构造的深度包检测方法，其特征在于，步骤 e 所述的基于后序遍历操作对二叉解析树进行节点编号的具体方法包括：

i. 初始化编号 n:=1；

ii. 若当前的节点是连接运算符节点，且该节点的前两个节点分别为闭包运算和连接运算，则，置该节点的整数标识为 n，否则，按照步骤 iii 中情况执行；

iii. 若当前的节点为连接运算符节点，并且不满足步骤 ii 中的情况，则 n:=n+1，置当前节点的整数标识为 n。

6.根据权利要求 2 所述的基于后缀自动机正则引擎构造的深度包检测方法，其特征在于，步骤 f 中所述的基于编号的二叉解析树设置自动机的终止状态标识的方法包括：

（1）若根节点为连接运算符节点，则将该节点的整数标识设置为终止状态标识；

（2）若二叉解析树的根节点是或运算节点，则置离根节点最近的左子树上的整数标识节点为终止状态节点；

（3）若二叉解析树的根节点是闭包运算节点，则不对任何一个整数标识置终止状态标识。

7. 根据权利要求 2 所述的基于后缀自动机正则引擎构造的深度包检测方法，其特征在于，步骤 g 所述的基于后缀正则树的中序遍历操作得后缀 NFA 引擎的方法包括：

1）假设后缀正则树中编号的最大整数为 n，则创建 n+1 个状态，状态名分别为 0、1、2、……、n，并将状态 0 设置为开始状态；

2）若后缀正则树中存在一个整数 i 被标识为终止状态，则将该整数 i 所对应的状态 i 设置为终止状态；若不存在，则将开始状态 0 同时设置为终止状态；

3）中序遍历后缀正则树，按照遍历顺序建立状态转移，最终获得只有一个开始状态和一个终止状态的并且 size 达到最优的 NFA。

基于后缀自动机正则引擎构造的深度包检测方法

技术领域

[0001] 本发明涉及一种基于后缀自动机正则引擎构造的深度包检测方法，属于网络安全技术领域。

背景技术

[0002] 近年来专门针对应用层的攻击技术和攻击工具正逐渐替代以往针对网络层和传输层的攻击。其中在应用层实施拒绝服务攻击，如基于 XML 的拒绝服务攻击（eXtensible Markup Language-based Denial of Service：X-DoS）和基于 HTTP 的拒绝服务攻击（Hypertext Transfer Protocol-based Denial of Service：H-DoS）已经成为 Web 技术和云计算所面临的最严重安全威胁。一方面是因为这类攻击实现起来非常简单，另一方面则是因为这些攻击数据包是以正常的协议数据包在网络上传递的，看起来与正常的访问完全一样，使得常用的包过滤防火墙以及基于协议包头分析的 IDS/IPS 等技术无法对其进行检测和防御。要想准确有效地实现对这类在应用层实现的拒绝服务攻击的检测，只有通过检测和分析应用业务数据流的内容来实现，因而诞生了深度包检测技术。

[0003] 深度包检测技术不仅检测协议数据包头部，而且深入到应用层业务数据包的有效载荷的内容中，分析和检测攻击行为的存在。以往的检测系统采用基于字符串的多模式匹配算法，典型的如 Aho-Corasick（简称 AC）算法及其改进算法；随着深度包检测技术的发展及被检测内容的日益复杂，基于表达能力强大灵活的正则表达式的多模式匹配技术逐渐取代了传统的基于字符串的精确模式匹配技术。Snort（一个开源的入侵检测系统）、Bro（一个轻量级的入侵检测系统）等入侵检测系统都将正则表达式应用到其规则集中；在协议分析领域中，传统的采用端口进行协议识别的方式也因其灵活性不够而被基于正则表达式的协议识别所代替，L7-filter（Linux Application Protocol Classifier，Linux 平台上的流量类型分类系统）、3Com 公司的 Tippingpoint X505 等系统也使用了基于正则表达式的多模式匹配算法来实现对应用层数据包的识别和安全检测。

[0004] 正则表达式在多模式时比精确字符串表现得更优异，在入侵检测协议分析、深度包检测等领域得到了广泛的应用。在基于正则表达式的深度包检测系统中，采用正则表达式实现的模式匹配主要是用有限自动机（通常称为正则引擎）来实现的。正则引擎是深度包检测系统中的模式匹配执行部件，也是其核心部件。根据所采用的自动机的不同，正则引擎分为 NFA（Non-deterministic Finite Automata，非确定型有限自动机）引擎、DFA（Deterministic Finite Automata，确定型有限自动机）引擎和 NFA/DFA 混合引擎。目前使用 DFA 引擎的程序主要有：awk，egrep，flex，lex，MySQL，Procmail 等；使用传统型 NFA 引擎的程序主要有：GNU Emacs，Java，ergp，less，more，.NET 语言，PCRE library，Perl，PHP，Python，Ruby，sed，vi；使用 POSIX NFA 引擎的程序主要有：mawk，Mortice Kern Systems' utilities，GNU Emacs（使用时可以明确指定）；也有使用 DFA/NFA 混合的引擎，如 GNU awk，GNU grep/egrep，Tcl。

[0005] 模式匹配的效率很大程度上取决于正则引擎的效率，包括正则引擎本身的规模

(size)和其运行效率。这里的size是指正则引擎所占用的存储空间的大小,由两个因素共同决定:一是自动机的状态数目,另一个是状态与状态之间的弧转移的数目。正则引擎(即自动机)可以用硬件方式实现的,也可以用软件方式实现的。典型的硬件方式实现的正则引擎(如申请号为200910238673.X、名称为基于正则表达式的深度包检测方法及设备的专利申请)是采用现场可编程门整列(Field-Programmable Gate Array,简称FPGA)来实现的。硬件NFA利用硬件的高速处理来加速多模式匹配的效率,弥补NFA计算复杂度高的缺陷。然而在实际应用中规则模式集合往往十分庞大并且在不断增加,类似基于FPGA的正则引擎的主频会随着字符组宽度的增大而迅速下降,并且由于其存储空间受限,只能存储少量的正则表达式,随着模式规则集中正则表达式的增多,必须相应地增加大量的硬件逻辑结构,再加上一个典型的模式集合往往由上百个正则表达式组成,因此硬件NFA引擎的扩展能力及其应用场合相当有限。

[0006] 在规则集是由大量的正则表达式组成的系统中,如Snort规则集中的正则表达式已经超过了3000条,通常都是采用软件方式来实现的。在NFA正则引擎中,NFA的size虽然与正则表达式的长度呈线性关系,但一个NFA只能对应一条正则表达式。因此,要实现多条正则表达式的同时匹配就需要由多个NFA并行执行来完成,而多个NFA的并行执行会导致匹配时所占用的内存达到几百兆甚至是几G且系统性能呈指数级急剧下降,严重地影响了检测效率。DFA正则引擎可以为多条正则表达式构造一个单一的DFA从而实现一个DFA对多条正则表达式的同时匹配,匹配效率很高,DFA引擎每完成一个字符的匹配只需要维护一个确定的状态转移,匹配算法的时间复杂度与模式串长度成正比,空间复杂度是长了。但是,构造多条正则表达式的单一DFA却存在空间爆炸问题,随着整合的正则表达式的数量增多,DFA的状态空间呈指数级急剧增长,导致存储代价巨大。在Cisco的安全产品中已达到GB级。可见,DFA引擎难以支持较多正则表达式。目前解决DFA空间爆炸问题的方案主要有以下几种:一种是对系统中的正则表达式进行分组,尽可能地避免构造多条正则表达式的DFA引起的空间爆炸,但这种分组方案非常有限并且扩展性极差;第二种是适当改写正则表达式以提高存储效率,但这种方案只能对特定形式的正则表达式进行改写,并不能改写任何形式的正则表达式,其适用范围非常有限;第三种通过挖掘状态转移的特点,采用延迟转移等方法来节省存储空间,但这种方案的匹配速度非常慢,严重地影响了系统的响应时间。

[0007] 最后,目前能够用于自动机构造的方法有四种:最早由Thompson提出的Thompson automata构造方法、由Glushkov提出的基于位置自动机(position automata)的Glushkov方法、由Antimirov提出的部分派生自动机(partial derivatives automata)和由Ilie和YU提出的后跟自动机(follow automata)。当前实际应用系统中常用的构造NFA的方法是Thompson方法和Glushkov方法两种。Thompson方法首先为每一条单独的正则表达式创建一个自动机,然后利用 ε 变换(空弧转换)将多个小自动机(称为自动机片段)合并到一起,包括连接、重复和或操作,最终拼接成一个完整的NFA。Glushkov构造方法的状态数是固定的,为m+1(其中m为正则表达式中输入符号的数目)。Glushkov方法首先将每个字符在表达式中的位置标记出来,标记了位置的正则表达式记为R',每个字符的位置表示经过该字符后到达的状态号,然后基于这种位置标记来构造自动机,最后消除所有的位置索引,最终抽取出Glushkov自动机。Glushkov自动机是基于位置自动机(position automata)理论的。

通常，实际系统中的DFA的构造是建立在NFA的基础上的，也就是首先构造Thompson NFA或Gloshkov NFA，然后再转换为DFA。Thompson方法本身简单且易于理解和实现，但是该方法在构造过程中引入了大量的空弧（ε-transition），导致了大量的无效的空状态和空弧转换。Position automata理论能够得到不含空弧转换的NFA，然而其构造方法存在两个方面的不足，一是其所获得的不带空弧转换的NFA的规模是非线性的，确切地说是在O(n)和O（n^2）之间；另一是其本身的实现繁杂，其实现过程需要经过多个处理过程，难于理解和实现。Partial derivatives automata构造算法能够获得较Thompson和Glushkov的规模更小的自动机，但是其构造实现过程本身非常复杂，其时间复杂度为O(n^5)，其所获得的正则引擎本身的时间复杂度为O(n log 2(n))，同样，这个方法也是建立在大量的集合计算的基础之上的，难于理解和编码实现。基于Follow automata理论所构造得到的NFA的size比上述4种都要小，然而该方法是基于两个算法来实现的，其实现过程本身很繁杂，不易理解和实现，并且所获得的NFA的size依然没有达到最优，且其Follow集的计算繁琐且不易实现。

[0008] 由上文可知，现有的基于正则表达式的深度包检测技术主要存在以下缺点：

[0009] 首先，采用硬正则引擎的深度包检测由于硬件本身的资源有限，无法满足大量正则表达式的应用需要，且引擎的主频会随着字符组宽度的增大而迅速下降；

[0010] 再者，采用软正则引擎的深度包检测系统的两种实现方式中，NFA引擎存在一个NFA只能对应一条正则表达式，多个NFA的并行执行会导致匹配时所占用的内存达到几百兆甚至是几G且系统性能呈指数级急剧下降的问题；DFA引擎则存在空间爆炸问题，随着整合的正则表达式的数量增多，DFA的状态空间呈指数级急剧增长，检测效率因此急剧降低；

[0011] 最后，目前所有现有的方法构造所得到的正则NFA引擎的规模都不是最优（size不是最小）的，而且除了Thompson方法之外，其它三种方法均需要完成大量抽象的包括计算follow集合在内的数学计算，在实际应用系统中难以编码实现。

[0012] 因此，需要一种有效的解决方案，能够在有效地解决NFA不能实现多条正则表达式匹配和DFA状态空间膨胀问题的同时，以更高的效率和更小的开销来实现基于正则表达式的深度包检测中的多模式匹配。

发明内容

[0013] 本发明的目的在于，提供一种基于后缀自动机正则引擎构造的深度包检测方法，它可以有效解决现有技术中存在的问题，尤其是基于正则表达式的多模式匹配技术中NFA引擎并行执行时性能急剧下降以及DFA引擎的空间爆炸的问题。

[0014] 为解决上述技术问题，本发明采用如下的技术方案：一种基于后缀自动机正则引擎构造的深度包检测方法，包括以下步骤：

[0015] S1，入侵检测系统抽取攻击特征构建正则表达式；

[0016] S2，入侵检测系统构造后缀NFA引擎并利用其进行多模式匹配；

[0017] S3，入侵检测系统从Web服务器获取应用层协议数据包和Web服务器日志文件；

[0018] S4，入侵检测系统对上述协议数据包和日志文件进行深度包检测并将检测结果发送给防火墙。

[0019] 上述的后缀自动机是指采用本发明后，基于正则表达式的后缀式的解析树所构造的只有一个开始状态和一个终止状态的不确定性有限自动机。

[0020] 步骤 S2 中所述的构造后缀 NFA 引擎的具体方法包括：

[0021] a. 对正则表达式进行分组；

[0022] b. 将每一组中的多条正则表达式利用或运算整合成一条正则表达式；

[0023] c. 将整合后的正则表达式改写成逆波兰形式，得后缀正则式；

[0024] d. 构造后缀正则式的二叉解析树；

[0025] e. 基于后序遍历操作对二叉解析树进行节点编号，确定自动机的状态空间；

[0026] f. 基于编号的二叉解析树设置自动机的终止状态标识，得后缀正则树；

[0027] g. 基于后缀正则树的中序遍历操作得后缀 NFA 引擎。

[0028] 上述步骤 c 中所述的将整合后的正则表达式改写成逆波兰形式，得后缀正则式的具体方法包括：

[0029] A. 正则表达式中按照原有顺序依次出现的输入符号作为后缀正则式从左至右输入的符号；

[0030] B. 运算符号按其实际计算顺序紧跟在其运算对象后面出现；

[0031] C. 将正则表达式中的小括号删除。

[0032] 上述步骤 d 中所述的构造后缀正则式的二叉解析树的方法是：

[0033] 1）后缀正则式中的所有输入符号作为二叉解析树的叶子节点，从左至右依次排列；

[0034] 2）根据后缀正则式中运算符的逻辑顺序依次由叶子节点到根节点自底向上建立整棵二叉解析树，其中，或运算和连接运算为双目运算，其运算对象：输入符号所对应的叶子节点或某个运算的运算结果所对应的子树的根节点分别为其左右孩子；闭包运算为单目运算，其运算对象为其左孩子，右孩子空缺。

[0035] 前述步骤 e 所述的基于后序遍历操作对二叉解析树进行节点编号的具体方法包括：

[0036] i. 初始化编号 n:=1；

[0037] ii. 若当前的节点是连接运算符节点，且该节点的前两个节点分别为闭包运算和连接运算，则，置该节点的整数标识为 n，否则，按照步骤 iii 中情况执行；

[0038] iii. 若当前的节点为连接运算符节点，并且不满足步骤 ii 中的情况，则 n:=n+1，置当前节点的整数标识为 n。

[0039] 步骤 f 中所述的基于编号的二叉解析树设置自动机的终止状态标识的方法包括：

[0040] （1）若根节点为连接运算符节点，则将该节点的整数标识设置为终止状态标识；

[0041] （2）若二叉解析树的根节点是或运算节点，则置离根节点最近的左子树上的整数标识节点为终止状态节点；

[0042] （3）若二叉解析树的根节点是闭包运算节点，则不对任何一个整数标识置终止状态标识。

[0043] 称完成步骤 e 和 f 后得到的指标正则解析树为后缀正则树。

[0044] 步骤 g 所述的基于后缀正则树的中序遍历操作得后缀 NFA 引擎的方法包括：

[0045] 1)假设后缀正则树中编号的最大整数为 n，则创建 n+1 个状态，状态名分别为 0、1、

2、.....、n，并将状态 0 设置为开始状态；

[0046] 2）若后缀正则树中存在一个整数 i 被标识为终止状态，则将该整数 i 所对应的状态 i 设置为终止状态；若不存在，则将开始状态 0 同时设置为终止状态；

[0047] 3）中序遍历后缀正则树，按照遍历顺序建立状态转移，最终获得只有一个开始状态和一个终止状态的并且 size 达到最优的 NFA。

[0048] 称由上述操作过程获得的 NFA 引擎为后缀自动机。

[0049] 本发明所述的方法还包括：

[0050] S5. 进行 IP 溯源，追踪到攻击源后，将攻击源的 IP 地址发送到防火墙进行包过滤，即实现了快速、高效的入侵检测。

[0051] 与现有技术相比，本发明解决了如下几个技术难题：1、用 NFA 方式实现了 DFA 的用单一自动机完成对多条正则表达式的匹配，解决了 NFA 不能实现多正则表达式匹配和 DFA 在实现多正则表达式匹配时的空间爆炸问题；2、有效地缩减了 NFA 的空间规模，解决了传统的 NFA 引擎构造方法所存在的空间浪费以及在执行模式匹配过程中的无效遍历问题，有效地缩短了了深度包检测的响应时间，提高了系统的整体性能和效率。以下进行详细分析：

[0052] （1）本发明的构造方法中，通过利用或运算将多条正则表达式合并成一条正则表达式，然后在避免 DFA 引擎的状态爆炸问题的情况下，基于后缀自动机构造出多条正则表达式的单一 NFA 正则引擎，从而有效地解决了采用 NFA 正则引擎的深度包检测系统中，一个自动机只能实现一条正则表达式的匹配，多条正则表达式的并行匹配引起的性能指数级下降的问题。

[0053] （2）使用本发明的构造方法能够获得空间规模达到最优的正则引擎，所获得的 NFA 不但具有 size 比现有的四种理论或方法都要小的 NFA，而且所获得的 NFA 只有一个开始状态和一个终止状态，从而有效地减少了 NFA 正则引擎的规模，同时降低了深度包检测过程中正则表达式的模式匹配复杂度，使得正则引擎本身的空间效率得以大大的提高，进而显著地提升了入侵检测系统的模式匹配效率；

[0054] （3）本发明的构造方法中引入了基于逆波兰式的正则表达式的解析树的构造方法，该方法通过构造正则表达式的运算逻辑的后缀解析树。（注：这里的后缀解析树是基于正则表达式的运算序列的二叉解析树，而不是通常意义上的基于字符串的后缀子串的后缀树（suffix trie））在后缀解析树的基础上，根据正则语言集上各种同态运算的封闭性和正则性原理，采用了一种前所未有的有限状态空间中最少状态的识别和标识方法，基于该方法，能够简单地通过对后缀解析树的特殊节点进行编码而获得 NFA 的所有状态以及状态时间所有的状态转移关系。在完成对后缀解析树的编码和标识的基础上，进一步地利用二叉树遍历操作即可简单地构造出比现有的四种方法所获得的 size 都要小的已经达到最优状态的只有一个开始状态和一个结束终止状态的 NFA。从而简化了正则引擎的构造过程，使得入侵检测系统本身的设计和实现更加简单、高效。

[0055] 本发明的难度在于：

[0056] 1）基于形式语言学和自动机理论中正则集上的同态运算的封闭性和正则性原理发现并证明了直接基于正则表达式的逆波兰式的解析二叉树获得 NFA 引擎所对应的最小状态空间的方法和原理；

[0057] 2）形式语言学和自动机理论中正则集上的同态运算的封闭性和正则性原理直接基于正则后缀树获得状态空间中唯一一个终止状态的标识及其正确性保证；

[0058] 3）基于单一简单的二叉树遍历操作即可实现状态以及状态转移的构造，并保证所构造的 NFA 与原正则表达式的等价性。

附图说明

[0059] 图 1 是本发明的一种实施例的工作流程图；

[0060] 图 2 为采用 Thompson automata 方法构造 NFA 引擎的示意图；

[0061] 图 3 为采用 Position automata 方法构造 NFA 引擎的示意图；

[0062] 图 4 采用 Follow automata 方法构造 NFA 引擎的示意图；

[0063] 图 5 为采用 Partial automata 方法构造 NFA 引擎的示意图；

[0064] 图 6 为本发明采用后缀自动机方法构造 NFA 引擎的示意图；

[0065] 图 7 为构造后缀正则式的二叉解析树的第一步示意图；

[0066] 图 8 为构造后缀正则式的二叉解析树的第二步示意图；

[0067] 图 9 为构造后缀正则式的二叉解析树的第三步示意图；

[0068] 图 10 为构造后缀正则式的二叉解析树的第四步示意图；

[0069] 图 11 为构造后缀正则式的二叉解析树的第五步示意图；

[0070] 图 12 为构造后缀正则式的二叉解析树的第六步示意图；

[0071] 图 13 为构造后缀正则树的第一步示意图；

[0072] 图 14 为构造后缀正则树的第二步示意图；

[0073] 图 15 为构造后缀正则树的第三步示意图；

[0074] 图 16 为构造后缀正则树的第四步示意图；

[0075] 图 17 为构造后缀正则树的第五步示意图；

[0076] 图 18 为构造后缀正则树的第六步示意图；

[0077] 图 19 为根据后缀正则树构造后缀 NFA 引擎的第一步示意图；

[0078] 图 20 为根据后缀正则树构造后缀 NFA 引擎的第二步示意图；

[0079] 图 21 为根据后缀正则树构造后缀 NFA 引擎的第三步示意图；

[0080] 图 22 为根据后缀正则树构造后缀 NFA 引擎的第四步示意图；

[0081] 图 23 为根据后缀正则树构造后缀 NFA 引擎的第五步示意图；

[0082] 图 24 为根据后缀正则树构造后缀 NFA 引擎的第六步示意图。

[0083] 下面结合附图和具体实施方式对本发明作进一步的说明。

具体实施方式

[0084] 实施例 1：一种基于后缀自动机正则引擎构造的深度包检测方法，如图 1 所示，包括以下步骤：

[0085] S1，入侵检测系统抽取攻击特征构建正则表达式；

[0086] S2，构造后缀 NFA 引擎并利用其进行多模式匹配；

[0087] S3，从 Web 服务器获取应用层协议数据包和 Web 服务器日志文件；

[0088] S4，对上述协议数据包和日志文件进行深度包检测并将检测结果发送给防火墙；

[0089] S5. 进行 IP 溯源，追踪到攻击源后，将攻击源的 IP 地址发送到防火墙进行包过滤。

[0090] 步骤 S2 中所述的构造后缀 NFA 引擎的具体方法包括：

[0091] a. 对正则表达式进行分组；

[0092] b. 将每一组中的多条正则表达式利用或运算整合成一条正则表达式；

[0093] c. 将整合后的正则表达式改写成逆波兰形式，得后缀正则式；

[0094] d. 构造后缀正则式的二叉解析树；

[0095] e. 基于后序遍历操作对二叉解析树进行节点编号，确定自动机的状态空间；

[0096] f. 基于编号的二叉解析树设置自动机的终止状态标识，得后缀正则树；

[0097] g. 基于后缀正则树的中序遍历操作得后缀 NFA 引擎。

[0098] 步骤 c 中所述的将整合后的正则表达式改写成逆波兰形式，得后缀正则式的具体方法包括：

[0099] D. 正则表达式中按照原有顺序依次出现的输入符号作为后缀正则式从左至右输入的符号；

[0100] E. 运算符号按其实际计算顺序紧跟在其运算对象后面出现；

[0101] F. 将正则表达式中的小括号删除。

[0102] 步骤 d 中所述的构造后缀正则式的二叉解析树的方法是：

[0103] 1）后缀正则式中的所有输入符号作为二叉解析树的叶子节点，从左至右依次排列；

[0104] 2）根据后缀正则式中运算符的逻辑顺序依次由叶子节点到根节点自底向上建立整棵二叉解析树，其中，或运算和连接运算为双目运算，其运算对象：输入符号所对应的叶子节点或某个运算的运算结果所对应的子树的根节点分别为其左右孩子；闭包运算为单目运算，其运算对象为其左孩子，右孩子空缺。

[0105] 步骤 e 所述的基于后序遍历操作对二叉解析树进行节点编号的具体方法包括：

[0106] i. 初始化编号 n:=1；

[0107] ii. 若当前的节点是连接运算符节点，且该节点的前两个节点分别为闭包运算和连接运算，则，置该节点的整数标识为 n，否则，按照步骤 iii 中情况执行；

[0108] iii. 若当前的节点为连接运算符节点，并且不满足步骤 ii 中的情况，则 n:=n+1，置当前节点的整数标识为 n。

[0109] 步骤 f 中所述的基于编号的二叉解析树设置自动机的终止状态标识的方法包括：

[0110] （1）若根节点为连接运算符节点，则将该节点的整数标识设置为终止状态标识（如在图中用双圆圈表示）；

[0111] （2）若二叉解析树的根节点是或运算节点，则置离根节点最近的左子树上的整数标识节点为终止状态节点；

[0112] （3）若二叉解析树的根节点是闭包运算节点，则不对任何一个整数标识置终止状态标识。

[0113] 步骤 g 所述的基于后缀正则树的中序遍历操作得后缀 NFA 引擎的方法包括：

[0114] 1）假设后缀正则树中编号的最大整数为 n，则创建 n+1 个状态，状态名分别为 0、1、2、.....、n，并将状态 0 设置为开始状态；

[0115] 2）若后缀正则树中存在一个整数 i 被标识为终止状态，则将该整数 i 所对应的状

态 i 设置为终止状态；若不存在，则将开始状态 0 同时设置为终止状态；

[0116] 3）中序遍历后缀正则树，按照遍历顺序建立状态转移，最终获得只有一个开始状态和一个终止状态的并且 size 达到最优的 NFA。

[0117] 实施例 2：一种基于后缀自动机正则引擎构造的深度包检测方法，包括以下步骤：

[0118] S1，入侵检测系统抽取攻击特征构建正则表达式；

[0119] S2，构造后缀 NFA 引擎并利用其进行多模式匹配；

[0120] S3，从 Web 服务器获取应用层协议数据包和 Web 服务器日志文件；

[0121] S4，对上述协议数据包和日志文件进行深度包检测并将检测结果发送给防火墙。

[0122] 步骤 S2 中所述的构造后缀 NFA 引擎的具体方法包括：

[0123] a. 对正则表达式进行分组；

[0124] b. 将每一组中的多条正则表达式利用或运算整合成一条正则表达式；

[0125] c. 将整合后的正则表达式改写成逆波兰形式，得后缀正则式；

[0126] d. 构造后缀正则式的二叉解析树；

[0127] e. 基于后序遍历操作对二叉解析树进行节点编号，确定自动机的状态空间；

[0128] f. 基于编号的二叉解析树设置自动机的终止状态标识，得后缀正则树；

[0129] g. 基于后缀正则树的中序遍历操作得后缀 NFA 引擎。

[0130] 步骤 c 中所述的将整合后的正则表达式改写成逆波兰形式，得后缀正则式的具体方法包括：

[0131] G. 正则表达式中按照原有顺序依次出现的输入符号作为后缀正则式从左至右输入的符号；

[0132] H. 运算符号按其实际计算顺序紧跟在其运算对象后面出现；

[0133] I. 将正则表达式中的小括号删除。

[0134] 步骤 d 中所述的构造后缀正则式的二叉解析树的方法是：

[0135] 1）后缀正则式中的所有输入符号作为二叉解析树的叶子节点，从左至右依次排列；

[0136] 2）根据后缀正则式中运算符的逻辑顺序依次由叶子节点到根节点自底向上建立整棵二叉解析树，其中，或运算和连接运算为双目运算，其运算对象：输入符号所对应的叶子节点或某个运算的运算结果所对应的子树的根节点分别为其左右孩子；闭包运算为单目运算，其运算对象为其左孩子，右孩子空缺。

[0137] 步骤 e 所述的基于后序遍历操作对二叉解析树进行节点编号的具体方法包括：

[0138] i. 初始化编号 n:=1；

[0139] ii. 若当前的节点是连接运算符节点，且该节点的前两个节点分别为闭包运算和连接运算，则，置该节点的整数标识为 n，否则，按照步骤 iii 中情况执行；

[0140] iii. 若当前的节点为连接运算符节点，并且不满足步骤 ii 中的情况，则 n:=n+1，置当前节点的整数标识为 n。

[0141] 步骤 f 中所述的基于编号的二叉解析树设置自动机的终止状态标识的方法包括：

[0142] （1）若根节点为连接运算符节点，则将该节点的整数标识设置为终止状态标识；

[0143] （2）若二叉解析树的根节点是或运算节点，则置离根节点最近的左子树上的整数标识节点为终止状态节点；

[0144] （3）若二叉解析树的根节点是闭包运算节点，则不对任何一个整数标识置终止状态标识。

[0145] 步骤 g 所述的基于后缀正则树的中序遍历操作得后缀 NFA 引擎的方法包括：

[0146] 1)假设后缀正则树中编号的最大整数为 n，则创建 n+1 个状态，状态名分别为 0、1、2、.....、n，并将状态 0 设置为开始状态；

[0147] 2）若后缀正则树中存在一个整数 i 被标识为终止状态，则将该整数 i 所对应的状态 i 设置为终止状态；若不存在，则将开始状态 0 同时设置为终止状态；

[0148] 3）中序遍历后缀正则树，按照遍历顺序建立状态转移，最终获得只有一个开始状态和一个终止状态的并且 size 达到最优的 NFA。

[0149] 实验例：

[0150] 硬件正则引擎支持的正则表达式数量极其有限，不适合当前网络信息系统，特别是云计算系统的深度包检测应用。当前 Snort 等软引擎系统，单一 DFA 匹配多正则表达式存在空间爆炸问题，传统 NFA 引擎存在并行匹配性能急剧下降问题。并且，当前已有的构造正则引擎的理论或方法只有四种，且实际系统基本都是基于 Thompson 自动机来实现的。现有的自动机构造理论所获得的 NFA 引擎都不是最小的，现用一个经典的实例对比说明如下：

[0151] 采用本发明以及现有的各种构造 NFA 的方法实现同一正则表达式 r 的 NFA 构造，其中，

[0152] r=（a+b）（a*+ab*+b*）*

[0153] 在本实验例中，Thompson 自动机（如图 2 所示）的 size=8+14=22；位置自动机（Position automata）（如图 3 所示）的 size=7+22=29；后跟自动机（Follow automata）（如图 4 所示）的 size=4+8=12；部分派生自动机（Partial automata）（如图 5 所示）的 size=4+11=15；而本发明，后缀自动机（Postfix automata）（如图 6 所示）的 size=3+7=10。因此，在所有的正则 NFA 引擎的构造方法中，本发明的方法的 size 最小，达到了最优，其能够解决其他方法所不能解决的为正则表达式组构造单一 NFA 引擎的问题。

[0154] 此外，本发明根据形式语言学和自动机理论中正则集上的同态运算的封闭性和正则性原理进行了理论证明，证明了方法的正确性，然后采用 C 语言，在 Linux 操作系统上进行了编码实现，并用大量的实例，将其与开源系统 Snort 中的正则引擎进行了模拟对比分析，其每次得到的 NFA 的 size 都与理论证明相符，达到了最小（最优）。

[0155] 实例说明：

[0156] 为了说明本发明的技术方案，现用 6 个实例来说明其整个技术方案，这 6 个实例包含了正则表达式中的三个基本运算的所有可能的逻辑组合情况，如下：

[0157] （1）构造分组整合后的正则表达式的逆波兰式，如下所示，每个实例的第一行为正则表达式，第二行为其所对应的逆波兰式：

[0158] $$(1)\begin{cases} r=(a+b)(a*+ba*+b*)* \\ pr=ab+a*ba*\cdot+b*+*\cdot \end{cases}$$

[0159] $$(2)\begin{cases} r=a(ab*+ab*a)b \\ pr=aab*\cdot ab*\cdot a\cdot+\cdot b\cdot \end{cases}$$

[0160] $$(3)\begin{cases} r=ab(ab*ab+(ba*b)*+a)* \\ pr=ab\cdot ab*\cdot a\cdot b\cdot ba*\cdot b\cdot *+a+*\cdot \end{cases}$$

[0161] $$(4)\begin{cases} r=(abc)* \\ pr=ab\cdot c\cdot * \end{cases}$$

[0162] $$(5)\begin{cases} r=a(bc)*d+aab* \\ pr=abc\cdot c*\cdot d\cdot aa\cdot b*\cdot + \end{cases}$$

[0163] $$(6)\begin{cases} r=ab(a*b+b*ab+a) \\ pr=ab\cdot a*b\cdot b*a\cdot b\cdot +a+\cdot \end{cases}$$

[0164] （2）构造二叉解析树，如图 7 ～图 12 所示；

[0165] （3）执行方案中的步骤 e 和 f，得到后缀正则树，如图 13 ～图 18 所示；

[0166] 对正则解析树（二叉解析树）进行编码得到后缀正则树的方法为：后序遍历正则解析树，按照遍历顺序对连接运算的节点按照如下情况进行编码和置终止状态标志。首先按照如下方法对正则解析树的连接运算符所对应的节点进行编号：

[0167] 1）初始化编号 n:=1；

[0168] 2）若当前的节点是连接运算符节点，且该节点的前两个节点分别为闭包运算和连接运算，则，置该节点的整数标识为 n，否则，按照第 3）中情况执行；

[0169] 3）若当前结点为连接运算符结点，并且不满足第 2）种情况，则 n:=n+1，再置当前结点的整数标识为 n。

[0170] 接下来按照如下方法设置终止状态标识：

[0171] 1）若根节点为连接运算符节点，则将该节点的整数标识设置为终止状态标识（在图中用双圆圈表示）；

[0172] 2）若正则解析树的根节点是或运算结点，则置离根节点最近的左子树上的整数标识结点为终止状态节点；

[0173] 3）若正则解析树的根节点是闭包运算结点，则不对任何一个整数标识置终止状态标识。

[0174] （4）根据后缀正则树获得后缀 NFA 引擎，如图 19 ～图 24 所示。

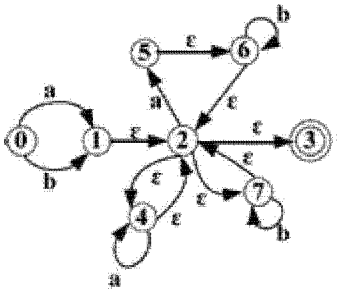

0
1
2
3
4
5
6
7
a
b
ε

图 2

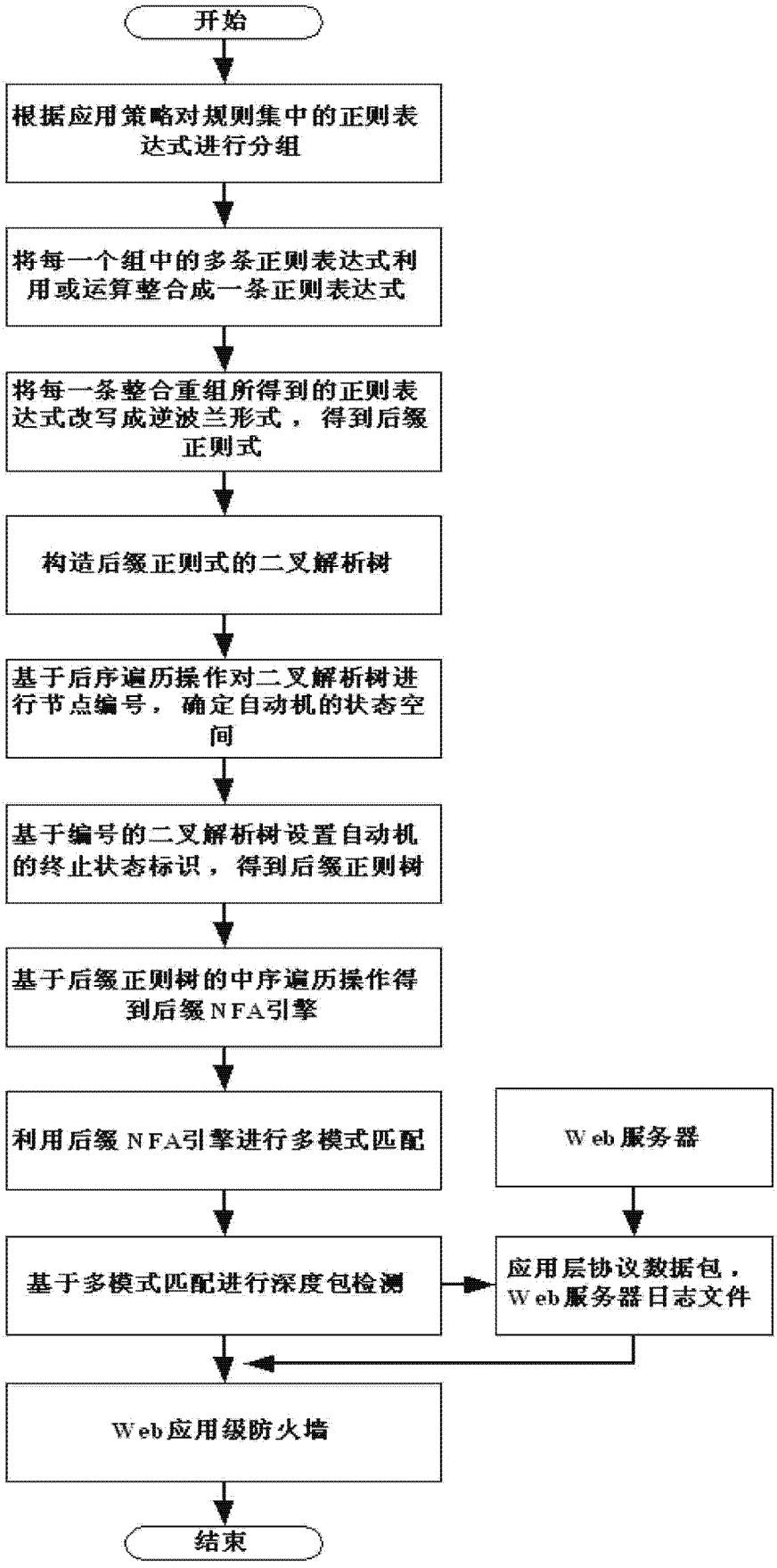

开始
根据应用策略对规则集中的正则表达式进行分组
将每一个组中的多条正则表达式利用或运算整合成一条正则表达式
将每一条整合重组所得到的正则表达式改写成逆波兰形式，得到后缀正则式
构造后缀正则式的二叉解析树
基于后序遍历操作对二叉解析树进行节点编号，确定自动机的状态空间
基于编号的二叉解析树设置自动机的终止状态标识，得到后缀正则树
基于后缀正则树的中序遍历操作得到后缀NFA引擎
利用后缀NFA引擎进行多模式匹配
Web服务器
基于多模式匹配进行深度包检测
应用层协议数据包，Web服务器日志文件
Web应用级防火墙
结束

图 1

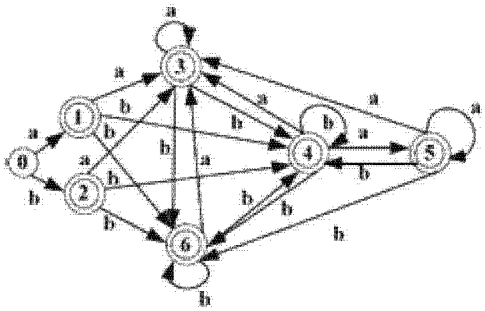


图 3

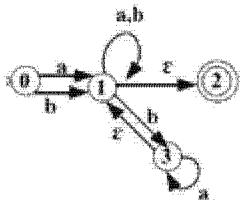


图 4

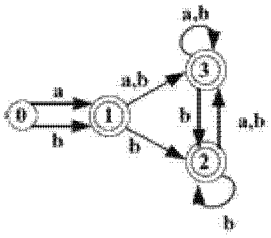


图 5

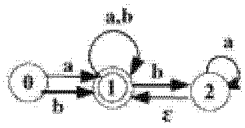


图 6

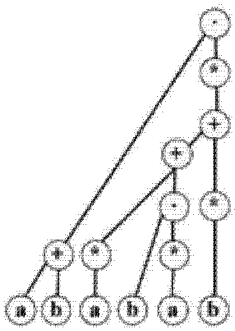


图 7

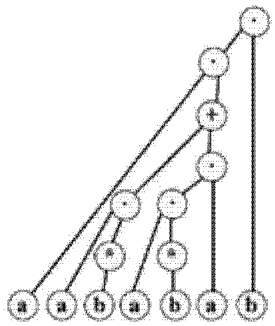


图 8

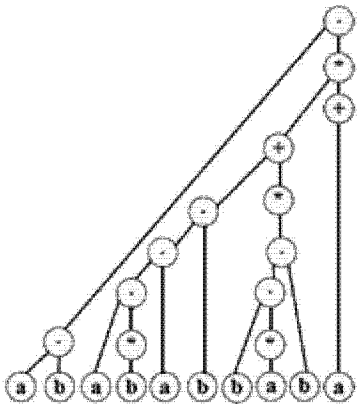


图 9

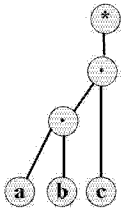


图 10

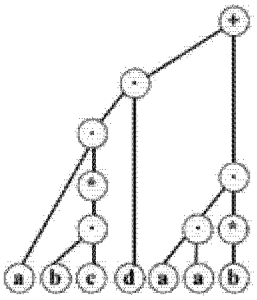


图 11

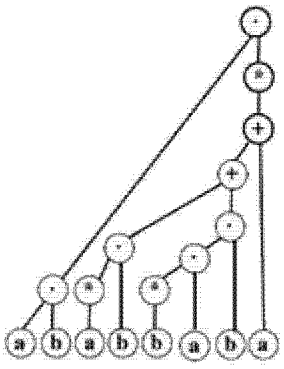


图 12

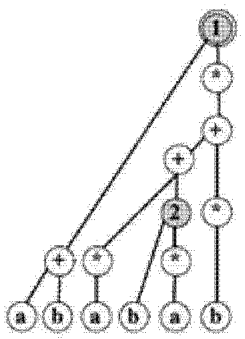


图 13

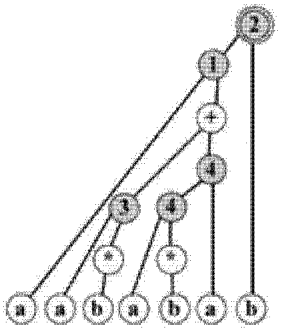


图 14

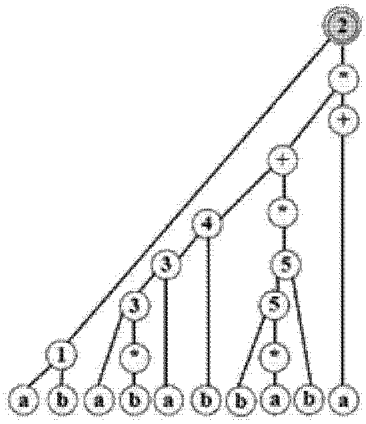


图 15

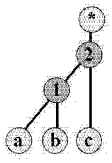


图 16

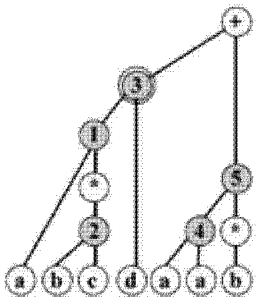


图 17

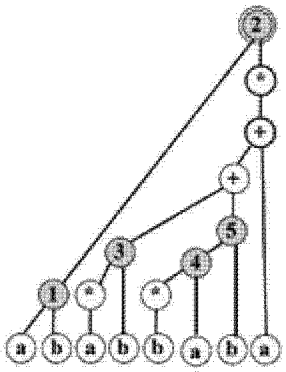


图 18

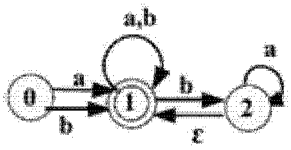


图 19

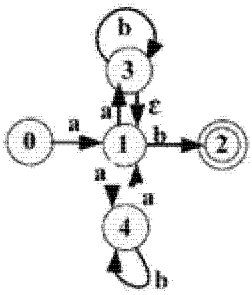


图 20

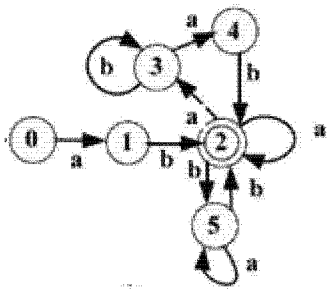


图 21

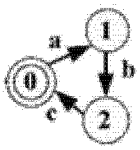


图 22

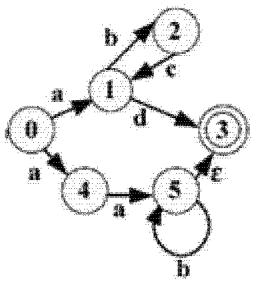


图 23

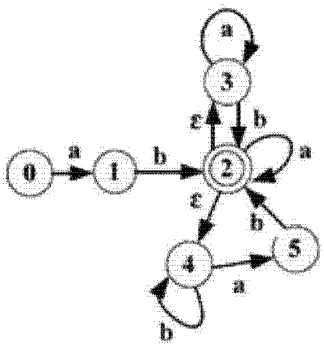


图 24