第8期 电 子 学 报 Vol.40 No.8
2012年8月 ACTA ELECTRONICA SINICA Aug. 2012

一种基于智能有限自动机的正则表达式匹配算法

张大方[1],张洁坤[1],黄 昆[2]
(1.湖南大学信息科学与工程学院,湖南长沙 410082; 2.中国科学院计算技术研究所,北京 100190)

摘 要: 本文提出了一种基于智能有限自动机(Smart Finite Automaton,SFA)的正则表达式匹配算法,在XFA的分支迁移边上增加额外的判断操作指令,消除XFA的回退迁移边,避免不必要的状态迁移操作.实验结果表明,SFA提高了正则表达式匹配的时空效率,与XFA相比,在存储空间开销上减少了44.1%,在存储器访问次数上减少了69.1%.

关键词: 深度数据包检测; 正则表达式匹配; 确定型有限自动机; 扩展有限自动机; 智能有限自动机
中图分类号: TP393.8 **文献标识码:** A **文章编号:** 0372-2112 (2012) 08-1617-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.08.019

A Regular Expression Matching Algorithm with Smart Finite Automaton

ZHANG Da-fang[1], ZHANG Jie-kun[1], HUANG Kun[2]
(1. *School of Information Science and Engineering, Hunan University, Hunan, Changsha* 410082, *China*;
2. *Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190, *China*)

Abstract: This paper presents a novel Regex matching algorithm with Smart Finite Automaton (SFA), where branching transitions of the XFA are augmented with adding extra check instruments, so that back-off transitions between states are eliminated, avoiding unnecessary state transitions. Experimental results show that compared with the XFA, the SFA significantly improves the time/space efficiency, separately reducing 44.1% and 69.1% in terms of the memory consumption and memory accesses of state transitions.

Key words: deep packet inspection; regular expression matching; deterministic finite automaton; extended finite automaton; smart finite automaton

1 引言

网络入侵检测与防御系统(Network Intrusion Detection and Prevention System, NIDS/NIPS)是网络安全防御的重要手段,通过实时监测网络流量,检查数据包的头部信息和有效载荷,从而识别和阻断可疑行为[1].NIDS/NIPS的核心是深度数据包检测(Deep Packet Inspection, DPI),即采用特征匹配算法,将每个数据包有效载荷与一组预定义的特征进行匹配.DPI技术不仅应用于NIDS/NIPS,而且还应用于应用层数据包分类、P2P流量识别、以及基于内容的流量计费等.

特征匹配算法可分为字符串匹配算法和正则表达式匹配算法.由于正则表达式具有丰富灵活的表达能力,可描述复杂特征,因而目前主流NIDS/NIPS例如Snort[2]、Bro[3]、TippingPoint IPS、Cicso IOS IPS等,已采用正则表达式匹配算法进行特征匹配.正则表达式匹配算法通常采用有限自动机来表示一组特征正则表达式.有限自动机可分为确定型有限自动机(Deterministic Finite Automata, DFA)和非确定型有限自动机(Nondeterministic Finite Automata, NFA).DFA具有时间高效等优点,但存在存储空间开销大等缺点;而NFA具有存储空间高效等优点,但存在匹配速度慢等缺点.因此,正则表达式匹配算法的关键是设计一种时空高效的有限自动机.

随着网络带宽和业务流量的迅猛增长,正则表达式匹配算法面临高性能挑战.一方面,线速数据包处理要求高吞吐量正则表达式匹配算法;另一方面,正则表达式匹配算法要求有限自动机存储在小容量快速存储器中(例如片上SRAM),进一步提高其吞吐量.Smith等人[4,5]提出一种基于扩展有限自动机(eXtended Finite Automaton, XFA)的正则表达式匹配算法,显著减少DFA存储空间需求.XFA是一种增强型DFA,即采用辅助变量替代额外DFA状态来记录部分匹配结果,执行简单

收稿日期:2011-03-11;修回日期:2012-04-12
基金项目:国家973重点基础研究发展计划(No.2012CB315805);国家自然科学基金(No.61173167,No.61100171);中国博士后科学基金(No.20100470023)

操作指令来检查匹配是否成功,从而消除 DFA 状态空间爆炸问题.与 DFA 相比,XFA 在状态空间上减少了 4 ~6 个数量级,而在匹配时间上接近 DFA.但是,XFA 存在冗余迁移边问题,不仅消耗更多存储空间,而且增加额外的存储器访问次数,从而限制了 XFA 性能.

为了解决上述问题,本文提出了一种基于智能有限自动机(Smart Finite Automaton,SFA)的正则表达式匹配算法,即在 XFA 的分支迁移边上增加额外的判断操作指令,减少 XFA 状态节点之间的冗余迁移边,从而提高正则表达式匹配的时空效率.实验结果表明,与 XFA 相比,SFA 在迁移边条数上减少了 56%,在存储空间开销上减少了 44.1%;在存储器访问次数上减少了 69.1%,在匹配时间上减少了 11%.

2 相关背景

本节首先介绍 DFA 状态空间爆炸问题,其次概述 XFA 的基本思想.

2.1 DFA 状态空间爆炸问题

采用 $D=(Q,\Sigma,\delta,q_0,F)$ 描述 DFA,其中 Q 是状态集合,Σ 是输入字母表集合,δ 是状态迁移函数,q_0 是起始状态,F 是接受状态集合,且 $F\subseteq Q$.当输入字符 a 时,对于任意状态 $q\in Q$,根据状态迁移函数 δ,D 迁移到下一状态 $q'=\delta(q,a)$.正则表达式的 DFA 构建过程:首先采用 Thompson 构造法[6]将正则表达式转化为 NFA;其次采用子集构造法将 NFA 转化为等价的 DFA.图 1 给出了正则表达式{.*ab.*cd}的单独 DFA,其中状态空间为{P,Q,R,S,T},字母表为 $\Sigma=\{a,b,c,d\}$.例如,起始状态为 $q_0=P$,状态迁移函数为 $\delta(P,a)=Q$、$\delta(Q,b)=R$ 等,而接受状态为 T.类似地,图 2 给出了正则表达式{.*ef.*gh}的单独 DFA,其中状态空间为{V,W,X,Y,Z},字母表为 $\Sigma=\{e,f,g,h\}$.

针对日益庞大的特征规则集,DPI 采用一组单独 DFA进行特征匹配,导致处理时间增加、匹配效率降低,因而采用一个联合 DFA 来表示一组特征规则.但是,联合 DFA 存在状态空间爆炸问题[5],即多个单独 DFA 的状态排列组合来记录部分匹配结果,导致整个 DFA难以存储在小容量的快速存储器中(例如片上 SRAM),需要频繁访问慢速存储器访问,从而降低正则表达式匹配算法的吞吐量.

图 3 给出了正则表达式{.*ab.*cd}和{.*ef.*gh}的联合 DFA.为了清晰阐述,图 3 省略了其他状态到初始状态的迁移边.如图 1 和图 2 所示,两个单独 DFA 共包含 10 个状态;而图 3 中的联合 DFA 包含 16 个状态.联合 DFA 的状态个数与正则表达式中通配符(例如“*”)的个数密切相关.由于通配符星号“*”表示任意字符,当采用一个联合 DFA 表示正则表达式{.*ab.*cd}和{.*ef.*gh}时,两个单独 DFA 的状态进行叉积组合,产生指数级增长的状态个数,造成联合 DFA 状态空间爆炸.

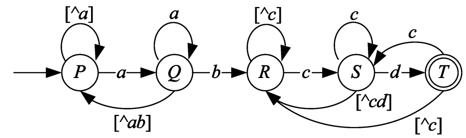


图1 正则表达式{.*ab.*cd}的单独DFA

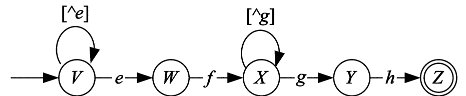


图2 正则表达式{.*ef.*gh}的单独DFA

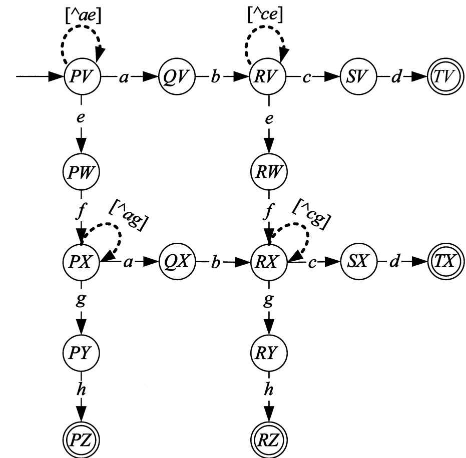


图3 正则表达式{.*ab.*cd}和{.*ef.*gh}的联合DFA

2.2 XFA 概述

为了消除联合 DFA 状态空间爆炸问题,Smith 等人[4,5]提出了一种扩展有限自动机(XFA),即采用辅助变量替代额外状态来记录部分匹配结果,执行简单操作指令来检查匹配是否成功.如果联合 DFA 的状态个数小于单独 DFA 的状态数之和,则每个单独 DFA 是非歧义的;反之,如果每个单独 DFA 是非歧义的,则联合 DFA 也是非歧义的.歧义的联合 DFA 采用歧义状态来记录部分匹配结果的排列组合.因此,针对歧义的联合 DFA,XFA 是在 DFA 状态上增加辅助比特变量,消除歧义状态,并在接受状态上执行比较指令,检查辅助比特变量是否设置.XFA 匹配过程是:当读入一个字符时,XFA 查找当前状态的相应迁移边,迁移到下一状态;执行下一状态的操作指令,通过检查辅助变量是否设置来判断匹配是否成功.

采用 $X=(Q,V,\Sigma,\delta,U,(q_0,v_0),F)$ 描述 XFA,其中 Q 是状态集合,V 是辅助变量集合,Σ 是输入字母表,$\delta:Q\times\Sigma\rightarrow Q$ 是状态迁移函数,$U:Q\times V\rightarrow V$ 是每个状态的更新函数,q_0 是起始状态,v_0 是辅助变量的初始值,$F\subseteq Q\times V$ 是接受状态集合.DFA 是根据当前状态和输入字符来决定下一个状态迁移,而 XFA 是根据

当前状态、辅助变量和输入字符来决定下一个状态迁移,并更新相应的辅助变量.

对于图 1 和图 2 的正则表达式{.*ab.*cd}和{.*ef.*gh},两个单独 DFA 的状态个数之和为 10,而图 3 的联合 DFA 的状态个数为 16,因而联合 DFA 是歧义的.针对正则表达式{.*ab.*cd},XFA 采用一个比特辅助变量 *Bit*1 来记录部分匹配结果 *ab*;针对正则表达式{.*ef.*gh},XFA 采用另一个比特辅助变量 *Bit*2 来记录部分匹配结果 *ef*.图 4 给出了正则表达式{.*ab.*cd}和{.*ef.*gh}的联合 XFA,仅采用 9 个状态和 2 个比特辅助变量,与图 3 的联合 DFA 是等价的,并消除了 DFA 状态空间爆炸.

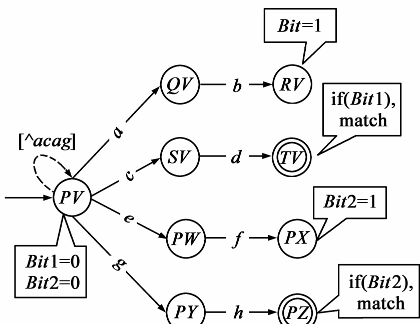


图4　正则表达式{.*ab.*cd}和{.*ef.*gh}的联合XFA

3　冗余迁移边问题

XFA 虽然消除了 DFA 状态空间爆炸问题,但是存在冗余迁移边问题,不仅导致存储空间需求大,而且增加存储器访问次数,从而限制了 XFA 的性能.为了简明指出 XFA 的冗余迁移边问题,图 5 和图 6 给出了匹配字符串 *abababcd* 和 *abcdabcdabcdefgh* 的 XFA 示例.

图 5 给出了正则表达式{.*ab.*cd}的 XFA,包含 5 个状态和 17 条迁移边.当读入字符串 *abababcd* 时,XFA 的状态迁移序列为 $P\to Q\to R\to Q\to R\to Q\to R\to S\to T$.当第 1 次到达状态 R 时,辅助变量 *Bit*1 设置为 1,表示等待到达状态 S 和 T,检查指令语句来指出匹配成功;由于 XFA 未记录到达状态 R,即已部分匹配字符串 *ab*,而等待后续字符串 *cd*,XFA 的迁移状态中 2 次出现重复状态 Q 和 R,导致不必要的状态迁移,从而产生额外的存储器访问次数和状态查找等开销.

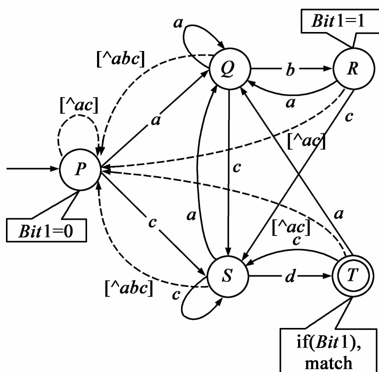


图5　正则表达式{*ab.*cd}的XFA

图 6 给出了正则表达式{.*abcd.*efgh}的 XFA,包含 9 个状态和 33 条迁移边.当读入字符串 *abcdabcdabcdefgh* 时,XFA 的状态迁移序列为 0→1→2→3→4→1→2→3→4→1→2→3→4→5→6→7→8,其中状态 1、2、3 和 4 重复出现 3 次.当第 1 次到达状态 3 时,辅助变量 *Bit*1 设置为 1,表示已部分匹配字符串 *abcd*,而等待后续字符串 *efgh*;由于 XFA 未判断辅助变量 *Bit*1 是否设置,XFA 执行不必要的状态迁移,导致额外的存储器访问次数和状态查找等开销.造成不必要的状态迁移的根本原因是 XFA 存在冗余失效迁移边(见图 5 和图 6 中所示的虚线迁移边),且未判断辅助变量是否设置.

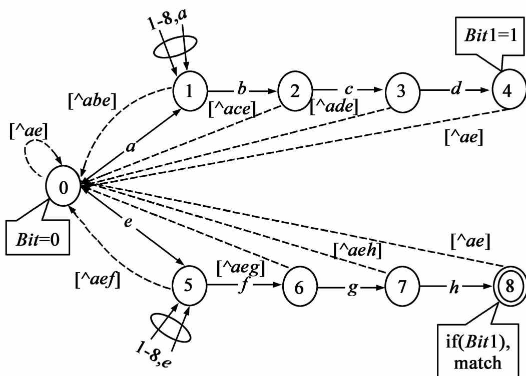


图6　正则表达式{.*abcd.*efgh}的XFA

4　智能有限自动机

为了解决 XFA 的冗余迁移边问题,本文提出了一种智能有限自动机(SFA),即在 XFA 基础上,在分支迁移边上增加辅助变量的判断指令,消除不必要的状态迁移,从而减少 XFA 存储空间开销并提高其匹配效率.SFA 的灵感来源是基于对 DFA 状态迁移的观察:如图 1 所示,当读入字符串 *abababcd* 时,DFA 的状态迁移序列为 $P\to Q\to R\to R\to R\to R\to R\to S\to T$,其中状态 R 重复出现 5 次.由于 DFA 的状态 R 具有记忆功能,即记录已部分匹配字符串 *ab*,等待后续字符串 *cd*;当读入非 *cd* 的字符串,DFA 始终迁移到状态 R,而不会回退到 R 之前的状态 P 或 Q.与 DFA 不同,XFA 虽然采用辅助变量和操作指令来消除 DFA 状态空间爆炸问题,但是却删除了状态的记忆功能.基于上述观察,本文的 SFA 是利用辅助变量,增加 XFA 迁移边的记忆功能,从而避免不必要的状态迁移.

SFA 的构建过程是:(1)在 XFA 的分支迁移边上增加操作指令来判断是否状态迁移,消除不必要的状态迁移;(2)消除 XFA 中回退迁移边,从而减少 XFA 的存储空间开销.其构建伪代码如下所示:

SFA 的构建伪代码

```
1:  XFA xfa;//给定一个 xfa
    //获取源状态为 q0 的所有迁移边
2:  ST = GetStateTransitions(xfa, q0);
    //在分支迁移边上增加判断操作指令
3:  for(each st in ST)
4:    if(st.DstState.Depth = = 1) then
5:      VariableSet + = {st.variable};//按序记录辅助变量
        //分支迁移边不指向 xfa 接受状态
6:      if(st.DstState.ForwardDirect! = AcceptState) then
7:        st·instruction = (if(! VariableSet)transit);
8:      else//分支迁移边指向 xfa 接受状态
9:        st.instruction = (if(VariableSet)transit);
10:     end
11:   end
12:end
13://消除回退迁移边
14:for(each q in Q)
15:  if(q! = q0) then
     //获取源状态为 q 的所有迁移边
16:    ST = GetStateTransitions(xfa, q);
17:    for(each st in ST)
18:      switch(q)
19:      {
20:      case MidState://中间状态
21:        if(st.DstState = = MidState) then
22:          delete st;
23:        end
24:        break;
25:      case PreState://前缀状态
26:        If((st.DstState = = MidState)and
           (In SameBranch(q, st.DstState) = = true))then
27:        delete st;
28:      end
29:      break;
30:    case AcceptState://接受状态
31:      If((st.DstState = = MidState)and
           (InSameBranch(q, st.DstState) = = false))then
32:        delete st;
33:      end
34:      break;
35:    }
36:  end
37: end
38:end
```

在 SFA 中,迁移边分为向前迁移边和交叉迁移边.向前迁移边是指从深度为 i 的节点指向深度为 $i+1$ 的节点的迁移边;而交叉迁移边是指从深度为 i 的节点指向深度为 j 的节点的迁移边,且 $i<=j$.分支迁移边是指从深度为 0 的节点指向深度为 1 的节点的向前迁移边.图 5 中的迁移边 $a:P\to Q$ 和 $c:P\to S$ 是分支迁移边.回退迁移边是指从中间状态 S_i 指向中间状态 S_j 的交叉迁移边,且 $i\neq j$,或者从前缀状态指向同一分支的中间状态的交叉迁移边,或者接受状态指向非同一分支的中间状态的交叉迁移边.前缀状态是指设置辅助变量为 1 的状态,而中间状态是除了初始状态、接受状态和前缀状态的其他状态.图 5 中的状态 R 是前缀状态、状态 Q 和 S 是中间状态,迁移边 $c:Q\to S$ 和 $a:S\to Q$ 是回退迁移边,而迁移边 $c:R\to S$ 不是回退迁移边.因此,在分支迁移边上,增加判断操作指令 if(*Bit*) transit,表示当辅助变量 *Bit* 设置为 1 时执行状态迁移,或者 if(! *Bit*) transit,表示当辅助变量 *Bit* 未设置为 1 时才执行状态迁移,可用于判断是否执行指定的状态迁移及查找下一个状态.当分支迁移边的判断操作指令表示不执行状态迁移时,SFA 保持当前状态不变,减少迁移边的存储空间访问次数,提高正则表达式匹配效率.

图 7 给出了正则表达式{.*ab.*cd}的 SFA,包含 5 个状态和 13 条迁移边.如图 7 所示,在分支迁移边 $a:P\to Q$ 上增加了判断操作指令 if(! *Bit*1) transit,在分支迁移边 $c:P\to S$ 上增加了判断操作指令 if(*Bit*1) transit,过滤掉不必要的状态迁移.与图 5 中 DFA 相比,SFA 具有相同的状态个数,但是其迁移边条数从 17 减至 13.图 8 给出了正则表达式{.*abcd.*efgh}的 SFA,包含 9 个状态和 25 条迁移边.如图 8 所示,在分支迁移边 $a:0\to1$ 和 $e:0\to5$ 上分别增加了相应的判断操作指令.与图 6 中 DFA 相比,SFA 的迁移边条数从 33 减至 25.第 5 节的实验结果表明:当正则表达式规则数量更多更复杂时,SFA 在少量迁移边上增加判断操作指令,但是其存储空间显著减少.

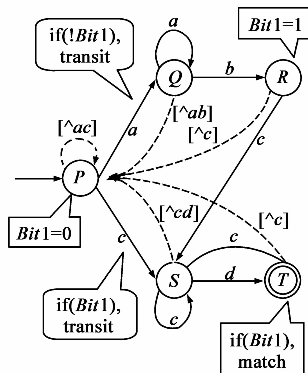


图7 正则表达式{.*ab.*cd}的SFA

当读入字符串 *ababab cd* 时,图 7 中 SFA 的匹配过程是:当读入字符 a 时,由于辅助变量 *Bit*1 设置为 0,初始状态 P 执行判断操作指令,迁移到状态 Q;当读入字符 b 时,状态 Q 迁移到前缀状态 R,并设置辅助变量 *Bit*1 为 1;当读入字符 a 时,状态 R 迁移到初始状态 P;当依次读入字符 b、a 和 b 时,由于辅助变量 *Bit*1 设置为 1,初始状态 P 执行判处操作指令,不迁移到其他状态,停留在初始状态 P;当读入字符 c 时,由于辅助变量 *Bit*1 设置为 1,初始状态 P 执行判处操作指令,迁移到状态 S;最后,当读入字符 d 时,状态 S 迁移到接受状态 T,并检查辅助变量是否设置为 1,匹配出正则表达式{.*ab.*cd}.因此,图 7 中 SFA 的状态迁移序列为 $P\to Q\to R\to P\to P\to P\to P\to S\to T$,其存储器访问次数为 5 次,少于 XFA 的 8 次.类似地,当读入字符串 *abcdabcdabcdefgh* 时,图 8 中 SFA 的状态迁移序列为 $0\to1\to2\to3\to4\to0\to0\to0\to0\to0\to0\to0\to0\to5\to6\to7\to8$,其存储器访问次数为 9 次,少于 XFA 的 16 次.

定理 1 SFA 与 XFA 是等价的,其中 SFA 为 $X'=$

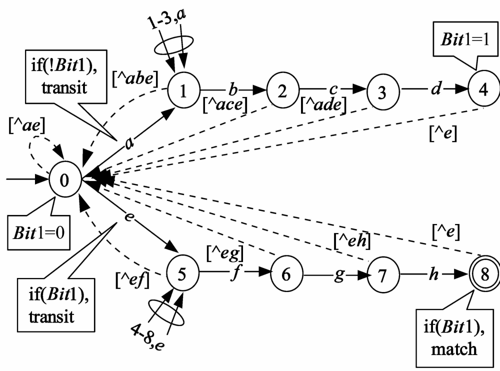


图8　正则表达式{.*abcd.*efgh}的SFA

$(Q, V', \Sigma, \delta', U', (q_0, v_0), F)$,XFA 为 $X=(Q, V, \Sigma, \delta, U, (q_0, v_0), F)$.

证明　由于 SFA 和 XFA 具有相同状态集合 Q,假设 SFA 和 XFA 的源状态 $q_i \in Q$,读入字符$\partial \in \sum$.在 SFA 中,存在一条状态迁移边$\partial: q_i \rightarrow q_j'$;而在 XFA 中,存在一条状态迁移边$\partial: q_i \rightarrow q_j$.

如 q_i 为初始状态 q_0,SFA 中的 q_i 执行判断操作指令来决定是否状态迁移,当辅助变量 *Bit* 设置为 1,则设置该变量的分支迁移边不执行;而 XFA 中的 q_i 迁移到下一状态 q_j',沿着分支继续匹配,但因 *Bit* 为 1,在该分支上无法匹配成功,其结果与 SFA 中的迁移结果相同.

如 q_i 为前缀状态,q_i 设置了相应辅助变量 *Bit'* 为 1,则 SFA 中的 q_i 迁移到初始状态 q_0,而 XFA 中的 q_i 迁移到同一分支的下一状态 q_j,根据上述初始状态的后续迁移结果与 XFA 中 q_j 迁移结果是相同的;根据其构造过程,SFA 中的 q_i 迁移到不同分支的下一状态 q_j' 与 XFA 中的迁移到的下一状态 q_j 相同.

如果 q_i 为接受状态,根据其构造过程,SFA 中的 q_i 迁移到同一分支的下一状态 q_j' 与 XFA 中的迁移到的下一状态 q_j 相同;SFA 中的 q_i 迁移到初始状态 q_0,而 XFA 中的 q_i 迁移到不同分支的下一状态 q_j,根据上述初始状态的后续迁移结果与 XFA 中的 q_j 迁移结果也是相同的.

如果 q_i 为中间状态,XFA 中的 q_i 迁移到的下一状态 q_j 为中间状态,而 SFA 中的 q_i 迁移到初始状态 q_0,根据上述初始状态的后续迁移结果与 XFA 中的 q_j 迁移结果是相同的;XFA 中的 q_i 迁移到的下一状态 q_j 为非中间状态,根据其构造过程,SFA 中的 q_i 迁移到的下一状态 q_j' 与 XFA 中的 q_j 相同.

因此,SFA 中的状态迁移边$\partial: q_i \rightarrow q_j'$ 与 XFA 中的状态迁移边$\partial: q_i \rightarrow q_j$ 是等价的,则 SFA 与 XFA 是等价的.

证明完毕.

5　仿真实验评估

本文采用 C/C++ 设计和实现了 XFA 和 SFA,并运行在 CPU 为 Intel Celeron CPU 1.3GHZ、内存为 512MB 的计算机上.在软件模拟实验中,本文在不同通配符星号"∗"个数和不同子串长度的条件下评估正则表达式匹配算法的时空效率.空间效率指标包括状态个数、迁移边条数、指令条数和存储空间开销等;而时间效率指标包括状态迁移次数和匹配时间等.评估数据集包含 100 个规则集,且每个规则集包含 100 条形式为{.∗*SubStr*1.∗*SubStr*2….∗*SubStrN*}的正则表达式,其中子串 *SubStr*1、*SubStr*2 和 *SubStrN* 具有相同的长度;而测试数据集为 1MB 的字符串集.

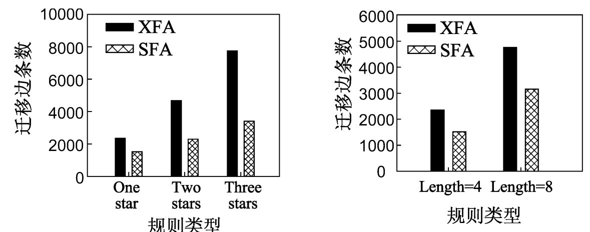
表 1 给出了在不同通配符个数和不同子串长度的条件下 *XFA* 和 *SFA* 的状态个数.表 1 显示,*SFA* 与 *XFA* 具有相同状态个数.

表 1　XFA 和 SFA 的状态个数

规则类型	状态个数	
	XFA	SFA
1 个通配符星号	897	897
1 个通配符星号	1300	1300
1 个通配符星号	1694	1694
子串长度为 4	897	897
子串长度为 8	1700	1700

5.1　空间效率

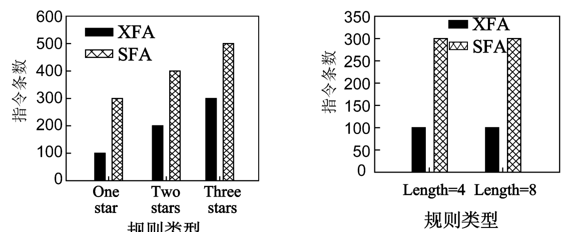
图 9 给出了 SFA 和 XFA 的迁移边条数比较.与 XFA 相比,SFA 在迁移边条数上减少了 56%;随着星号个数或子串长度的增加,SFA 减少的冗余迁移边条数比率也增多,即从 33.8%增至 56%.



(*a*) 相同子串长度且不同通配符个数　(*b*) 相同通配符个数且不同子串长度

图9　SFA和XFA的迁移边条数比较

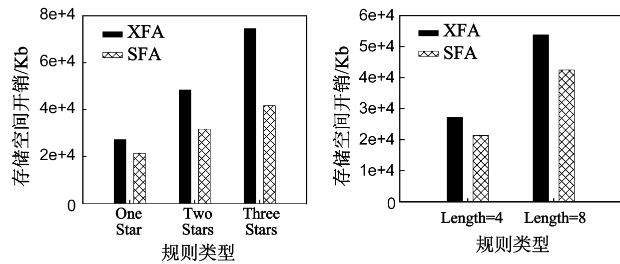
图 10 给出了 SFA 和 XFA 的指令条数比较.SFA 和 XFA 均采用相同的操作指令来执行状态迁移或匹配检查等,目前专用硬件(例如 GPU/SIMD)指令可支持.图 10 表明,与 XFA 相比,由于在分支迁移边上增加了额外的判断操作指令,SFA 的指令条数增多;另外,随着星号个数的增加,SFA 的指令条数也增加;但是,随着子串长度的增加,SFA 的指令条数保持恒定不变.



(*a*) 相同子串长度且不同通配符个数　(*b*) 相同通配符个数且不同子串个数

图10　SFA和XFA的指令条数比较

存储空间开销是由状态个数、迁移边条数、操作指令条数和辅助变量个数等决定的.在相同的实验条件下,SFA 与 XFA 的状态个数和辅助变量个数是相同的;SFA 的迁移边条数少于 XFA,而 SFA 的操作指令条数多于 XFA.因而,存储空间开销是 SFA 空间效率的关键指标.图 11 给出了 SFA 和 XFA 的存储空间开销比较.与 XFA 相比,SFA 在存储空间开销上减少了 44.1%;随着星号个数或子串个数的增加,SFA 减少的存储空间开销也增多,即从 21%增至 44.1%.

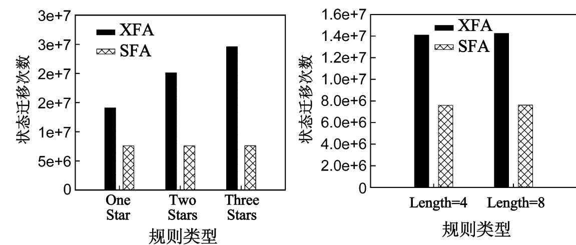


(*a*) 相同子串长度且不同通配符个数 (*b*) 相同通配符个数且不同子串个数

图11 SFA和XFA的存储空间开销比较

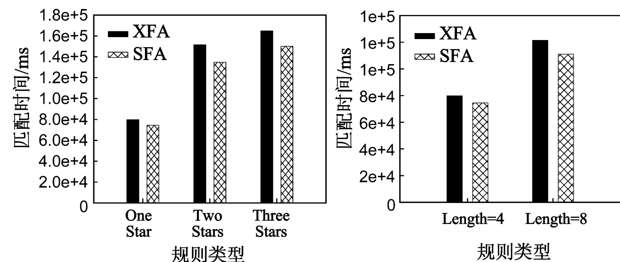
5.2 时间效率

状态迁移次数主要反映正则表达式匹配算法的存储器访问次数,即存储器带宽需求.由于高速存储器带宽受限和代价昂贵,减少状态迁移次数有助于减少存储器带宽需求,从而提高正则表达式匹配的性能以及减少其硬件开销.图 12 给出了 SFA 和 XFA 的状态迁移次数比较.与 XFA 相比,SFA 在状态迁移次数上减少了 46.2% ~ 69.1%.



(*a*) 相同子串长度且不同通配符个数 (*b*) 相同通配符个数且不同子串个数

图12 SFA和XFA的状态迁移次数比较



(*a*) 相同子串长度且不同通配符个数 (*b*) 相同通配符个数且不同子串个数

图13 SFA和XFA的匹配时间比较

本文是在相同硬件平台和评估数据集的条件下,统计 SFA 和 XFA 的实际匹配时间.由于受通用硬件平台的 CPU、I/O 总线带宽等限制,基于软件实现的 SFA 和 XFA 难以满足实际 10Gbps 线速数据包处理.但是,本文的匹配时间是在相同条件下的仿真匹配时间,可满足 SFA 和 XFA 的性能比较需求.图 13 给出了 SFA 和 XFA 的匹配时间比较.与 XFA 相比,SFA 在匹配时间上减少了 6.7% ~ 11%.

6 相关工作

Aho-Corasick[6] 和 Commentz-Walter[7] 等经典字符串匹配算法主要应用于早期的 NIDS/NIPS.这些基于软件的特征匹配算法难以满足高速数据包处理的吞吐量需求,因此研究者提出基于硬件的特征匹配算法,实现线速数据包内容过滤.特征匹配算法采用字符串难以准确描述复杂攻击特征;由于正则表达式具有丰富和灵活的表达能力,目前 NIDS/NIPS 采用正则表达式描述攻击特征以及相应的正则表达式匹配算法.但是,当特征规则条数不断增多时,正则表达式匹配算法面临 DFA 存储空间爆炸等问题,限制了正则表达式匹配的性能和可伸缩性.

近年来,研究者提出了多种存储高效的正则表达式匹配算法,减少 DFA 存储空间需求,并提高正则表达式匹配的吞吐量.Kumar 等人[8]提出了一种基于 D^2FA 的正则表达式匹配算法,即采用一条默认迁移边替代状态之间的多条相同迁移边,从而减少 DFA 迁移边条数,但是 D^2FA 存在匹配吞吐量低和默认迁移边构建开销高等缺点.Kumar 等人[9]又提出了一种基于 CD^2FA 的正则表达式匹配算法,即在状态上增加内容标识,用于记录下一状态的迁移边,从而避免不必要的状态迁移,提高 D^2FA 的吞吐量.Becchi 等人[10]提出了一种基于状态融合 DFA 的正则表达式匹配算法,即采用迁移边标记方法来融合多个非等价状态,减少 DFA 存储空间需求,并确保其最坏情况匹配性能.Zhang 等人[11]提出了一种基于迁移边融合 DFA 的正则表达式匹配算法,即融合具有相同目的状态的迁移边,采用优先级解析融合的迁移边,从而减少 DFA 存储空间开销.

为了消除 DFA 存储空间爆炸问题,Yu 等人[12]提出了一种基于 mDFA 的正则表达式匹配算法,即采用正则表达式重写和分组等启发式方法,将一个正则表达式规则集分割成多组规则子集,并采用多个 DFA 表示这些规则子集,从而减少整个规则集的存储空间需求,但是 mDFA 存在并行处理开销高等缺点.与 XFA 类似,Kumar 等人[13]采用启发式方法来消除正则表达式匹配算法的"失眠症",即 DFA 的不活跃状态占用许多片上存储空间,"健忘症",即 DFA 需要大量的额外状态来记录部分匹配结果,以及"失算症",即 DFA 无法记录相同子串的匹配次数.黄昆等人[14]提出了一种基于紧凑型有限自动机的正则表达式匹配算法,即采用基于优先

级的迁移边压缩方法，迭代压缩相同目的状态最多的迁移边；采用基于位图的迁移边查找方法，并行查找不同优先级的迁移边子集．

7　结论

本文指出已有的基于 XFA 的正则表达式匹配算法虽然消除了 DFA 状态空间爆炸问题，但是存在冗余迁移边问题，不仅导致存储空间需求大，而且产生不必要的状态迁移，限制了 XFA 的性能．本文提出了一种基于 SFA 的正则表达式匹配算法，即在 XFA 的分支迁移边上增加判断操作指令，避免了不必要的状态迁移，并消除 XFA 状态之间的回退迁移边，从而提高 XFA 的时空效率．实验结果表明：与 XFA 相比，SFA 在迁移边条数上减少了 56%，在存储空间开销上减少了 44.1%；在存储器访问次数上减少了 69.1%，在匹配时间上减少了 11%．因此，SFA 是一种快速和存储高效的有限自动机，可适用于高速数据包内容过滤和应用层流量识别等．

参考文献

[1] V Paxson，K Asanovic，S Dharmapurikar，et al．Rethinking hardware support for network analysis and intrusion prevention［A］．Proceedings of USENIX Workshop on Hot Topics in Security 2006［C］．Vancouver：USENIX Press，2006．

[2] M Roesch．Snort-lightweight intrusion detection for networks［A］．Proceedings of LISA 1999［C］．Seattle：USENIX Press，1999．

[3] V Paxson．Bro：A system for detecting network intruders in real-time［J］．Computer Networks，1999，31（23－24）：2435－2463．

[4] R Smith，C Estan，S Jha．XFA：Faster signature matching with extended automata［A］．Proceedings of IEEE Symposium on Security and Privacy 2008［C］．Oakland：IEEE Press，2008．

[5] R Smith，C Estan，S Jha，et al．Deflating the big bang：Fast and scalable deep packet inspection with extended finite automata［A］．Proceedings of ACM SIGCOMM 2008［C］．Seattle：ACM Press，2008．

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

[7] B Commentz-Walter．A string matching algorithm fast on the average［A］．Proceedings of 6th Colloquium on Automata，Languages and Programming［C］．London：Springer-Verlag Press，1979．

[8] S Kumar，S Dharmapurikar，F Yu，et al．Algorithms to accelerate multiple regular expressions matching for deep packet inspection［A］．Proceedings of ACM SIGCOMM 2006［C］．Pisa：ACM Press，2006．

[9] S Kumar，J Turner，J Williams．Advanced algorithms for fast and scalable deep packet inspection［A］．Proceedings of ACM/IEEE ANCS 2006［C］．San Jose：ACM Press，2006．

[10] M Becchi，S Cadambi．Memory-efficient regular expression search using state merging［A］．Proceedings of IEEE INFOCOM 2007［C］．Anchorage：IEEE Press，2007．

[11] J Zhang，D Zhang，K Huang．A regular expression matching algorithm using transition merging［A］．Proceedings of IEEE PRDC［C］．Shanghai：IEEE Press，2009．242－246．

[12] F Yu，Z Chen，Y Diao，et al．Fast and memory-efficient regular expression matching for deep packet inspection［A］．Proceedings of ACM/IEEE ANCS 2006［C］．San Jose：ACM Press，2006．

[13] S Kumar，B Chandrasekaran，J Turner．Curing regular expressions matching algorithms from insomnia，amnesia，and acalculia［A］．Proceedings of ACM/IEE ANCS 2007［C］．Orlando：IEEE Press，2007．155－164．

[14] 黄昆，张大方，谢高岗，金军航．一种面向深度数据包检测的紧凑型正则表达式匹配算法［J］．中国科学：信息科学，2010，40（2）：356－370．
Huang Kun，Zhang Da-fang，Xie Gao-gang，Jin Jun-hang．A compact regular expression matching algorithm for deep packet inspection［J］．SCIENCE CHINA Information Sciences，2010，40（2）：356－370．（in Chinese）

作者简介

张大方　男，1959 年出生于上海，湖南大学教授，博士生导师．主要研究方向为可信系统与网络、网络安全、网络测量等．
E-mail：dfzhang@hnu.edu.cn

张洁坤　女，1983 年出生于河北石家庄，湖南大学硕士毕业生．主要研究方向为网络安全．

黄　昆　男，1978 年出生于江西吉安，中国科学院计算技术研究所博士后．主要研究方向为网络安全、未来互联网等．
E-mail：huangkun09@ict.ac.cn

第8期 2012年8月
电 子 学 报
ACTA ELECTRONICA SINICA
Vol.40 No.8 Aug. 2012

本体匹配中基于词义组合的词法分析算法

刘秀磊[1,2],廖建新[1,2],朱晓民[1,2],杨 迪[1,2],徐 童[1,2]
(1.北京邮电大学网络与交换技术国家重点实验室,北京 100876;2.东信北邮信息技术有限公司,北京 100191)

摘 要: 针对本体匹配中相似性词法分析算法的不足,提出一种基于词义组合的词法分析算法.该算法首先通过 WordNet 发现本体中单词的合适词义,并扩展它们,然后基于本体里的语义元素形式化的定义实体的词法信息标记,最后推理出实体词法信息间的包含关系.针对一组工业本体的测试结果表明该算法有助提高系统的覆盖率.

关键词: 本体匹配; 词法分析; WordNet; 包含关系
中图分类号: TP182 **文献标识码:** A **文章编号:** 0372-2112 (2012) 08-1624-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.08.020

Lexical Analysis Based on Combining Senses in Ontology Matching

LIU Xiu-lei[1,2], LIAO Jian-xin[1,2], ZHU Xiao-min[1,2], YAND Di[1,2], XU Tong[1,2]
(1. *State Key Lab of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing* 100876, *China*;
2. *EBUPT Information Technology Ltd., Beijing* 100191, *China*)

Abstract: The paper presents a lexical analysis algorithm based on combining senses of words for computing subsumption relations between entities in ontology matching. It firstly finds the suitable sense of each word and extends it; then formally defines the representation of an entity notion based on semantic elements; finally, infers subsumption relations between all possible pair of entities, one from each of two ontologies. The experiments, over four real in use ontologies, show that the algorithm helps to increase the recall of the system.

Key words: ontology matching; lexical analysis; WordNet; subsumption

1 引言

随着本体使用的日益增长,表示相似(或相同)领域共享概念模型的大部分本体往往是由不同背景知识的工程师使用各种术语构造和维护.这些表示相似(或相同)领域的不同本体之间的异构性阻碍了系统对知识的共享、重用和互操作.本体匹配则是解决本体异构问题的方法之一.

目前大部分本体匹配系统[1~3]在分析本体的词法信息时多采用基于文本的相似性方法(比如文献[5~6])或基于词典的相似性方法(比如文献[7~8])计算不同本体中实体之间的词法相似性.通常这些相似性的值是在[0,1]之间的实数[4],不包含任何的语义关系,并且大部分本体匹配系统在采用相似性算法时忽视了以下问题:

• 词义组合性问题:实体标签和评论由多词汇构成,因此需考虑词汇间的关系.假设有两个概念标签 *Book* 和 *BookTitle*,尽管它们之间存在一定的相似性,但实际上 *Book* 与 *BookTitle* 表示不同的概念.再比如概念评论"*monograph or collection*"的词法信息表示的是"*monograph* ∪ *collection*".

• 词义模糊性问题:在词典里没有关系的词义,可以表达相同的概念.假如有两个属性 < *Book*, *publishedBy*, *IEEE* > 和 < *Book*, *hasPublisher*, *IEEE* >,尽管来自"*published*"和"*publisher*"的任何词义都不相关,但它们表示相同的角色,即出版商.

• 词义难寻问题:根据本体的上下文获得单词的合适词义在自然语言处理领域依然是个挑战.因此目前的词义发现技术会导致部分单词得到错误的词义.显然,得到错误的词义之后再计算它们之间的词义相似性是不可靠的.

本文提出的基于词义组合的词法分析算法,它不完全依靠单词的合适词义,通过使用 *WordNet* 里特定的词法关系,扩展单词的合适词义到一组词义以探索词义模

收稿日期:2012-01-28;修回日期:2012-07-10
基金项目:国家自然科学基金(No.61121001,No.61072057,No.60902051,No.61101119);长江学者和创新团队发展计划基金(No.IRT1049);国家科技重大专项基金(No.2011ZX02002-001-01,移动互联网总体架构研究)