计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200300096

自动机终结字查找算法的设计与实现

孙士远　何　勇
湖南科技大学计算机科学与工程学院　湖南 湘潭 411201
(2499644391@qq.com)

摘　要　自动机的秩与工业自动化中的部件定向器设计问题和理论计算机科学中的Černý-Pin猜想密切相关。计算自动机的秩可以归结于查找自动机的终结字。Rystsov于1992年提出了一个时间复杂度为$O(|A|^4)$的自动机终结字查找算法,该算法是至今仅有的专门用于计算自动机的终结字的算法。以现有同步自动机的同步字查找算法为蓝本可以设计几种自动机终结字查找的新算法。理论分析和实验结果表明,这些新算法都是Rystsov算法的优化。

关键词:终结字;同步对;同步字;Eppstein预处理
中图法分类号　TP301

Designs and Implementations of Algorithms for Searching Terminal Words of Automata

SUN Shi-yuan and HE Yong
School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, Hunan 411201, China

Abstract　The ranks of automata are strongly associated with the problem of the designs of parts orienters on industrial automation and Černý-Pin conjecture on theoretical computer science. Computing ranks of automata can be turned into computing terminal words of automata. Rystsov proposed an algorithm for searching terminal words of automata with $O(|A|^4)$ time complexity. The algorithm is the only one to compute terminal words of automata till now. Some new algorithms for searching terminal words of automata can be designed based on existing algorithms for searching synchronizing words of synchronizing automata. Theoretical analysis and experimental results show that all these new algorithms are optimizations of Rystsov algorithm.

Keywords　Terminal words, Synchronizing pair, Synchronizing words, Eppstein pre-processing

1　引言

用$A=(Q,\Sigma,\delta)$表示状态集为Q、输入字母表为Σ、状态转移函数为δ的(完全确定有限)自动机。Σ上的自由幺半群Σ^*就是由该自动机的所有输入字关于连接运算形成的代数系统。作为一个从$Q\times\Sigma$到Q的函数,状态转移函数δ可以自然地扩展为一个从$Q\times\Sigma^*$到Q的函数。由扩展后的状态转移函数δ定义的输入字w在状态q上的作用$\delta(q,w)$通常简单地记作qw。对于输入字w和状态集Q的子集P,定义:

$$Pw=\{pw:p\in P\},$$

$$Pw^{-1}=\{q\in Q:qw\in P\}$$

自动机A的秩定义为:

$$r(A)=\min_{w\in\Sigma^*}|Qw|。$$

如果A的输入字w具有性质$|Qw|=r(A)$,就称w为A的一个终结字,并称Qw为A的一个终结集。

Natarajan等考虑了工业自动化中的部件定向器设计问题[1-4]:给定一个部件O和O的方向的两个集合I和G,设计一个由一组静态障碍物构成的部件定向器S,O经过S后可以从I中的任意方向到达G中的某些方向。该问题可以用自动机的语言表述如下:对于自动机$A=(Q,\Sigma,\delta)$及其状态集的子集I和G,查找输入字w使$Iw\subseteq G$。显然,当$I=Q$时,部件定向器设计问题有解的必要条件之一是$|G|\geqslant r(A)$;特别地,当$I=Q$且$|G|=r(A)$时,部件定向器设计问题有解的充分必要条件是G为A的一个终结集。因此,部件定向器设计问题从实际应用上对自动机的秩的研究提出了一个明确的需求。

秩为1的自动机称为同步自动机,同步自动机的终结字称为同步字。关于同步自动机的研究[5-20]的核心问题是由Černý于1964年提出的Černý猜想:n状态同步自动机一定有长度不超过$(n-1)^2$的同步字。Černý猜想后来被一般化为Černý-Pin猜想:秩为r的n状态自动机一定有长度不超过$(n-r)^2$的终结字[21-22]。Černý-Pin猜想现在仍然是一个未解决的问题,它从理论上对自动机的秩的研究提出了又一个明确的需求。

自动机的秩的计算问题显然可以归结于自动机终结字的查找问题。Rystsov于1992年提出了一个时间复杂度为$O(|A|^4)$的自动机终结字查找算法[21],其中$|A|$表示自动机A的状态数和输入字母数的乘积。该算法涉及的理论基础比较复杂,计算的复杂度也偏高,不适合实际应用。本文以现有的同步自动机的同步字查找算法为蓝本[8,15],提出了几种自动机终结字的查找算法,从理论上分析了它们的复杂

基金项目:国家自然科学基金(61572013)
This work was supported by the National Natural Science Foundation of China (61572013).
通信作者:何勇(yonghe@hnust.edu.cn)

度，并用实验检测了它们的性能。

2 Rystsov 算法

设 $A=(Q,\Sigma,\delta)$是一个自动机。如果 A 的状态集 Q 的子集 X 在任意输入字母 a 的作用下都有 $Xa\subseteq X$，那么 $(X,\Sigma,\delta|_{X\times\Sigma})$也是一个自动机，其中 $\delta|_{X\times\Sigma}$表示 A 的状态转移函数 δ 在 $X\times\Sigma$ 上的限制。此时，称 X 为 A 的一个子自动机，记作 $X\leqslant A$。显然，A 的若干个子自动机的并集仍然是 A 的子自动机。进一步地，对于 A 的任意子自动机 X 都可以定义 A 模 X 的商自动机：

$$A/X=(Q\backslash X\cup\{x\},\Sigma,\delta_X)$$

其中，状态转移函数 δ_X 的定义如下：

$$\delta_X(p,a)=\begin{cases}\delta(p,a), & \text{如果 } p\neq x\\ x, & \text{如果 } p=x\end{cases}$$

对于自动机 A 的状态 p 和 q，如果 $p=q$ 或者存在输入字 $w\in\Sigma^*$ 使得 $pw=q$，就称 p 到 q 是可达的。如果 A 的每个状态到其他任意状态都是可达的，就称 A 是强连通的。自动机 A 的所有强连通子自动机的并集称为 A 的核，记作 $Nu(A)$，集合 $Co(A)=Q\backslash Nu(A)$称为 A 的余核。显然，$Nu(A)$是 A 的一个子自动机。如果 A 的输入字 w 满足条件 $Qw\subseteq Nu(A)$，就称 w 为 A 的一个核字。对于 A 的余核 $Co(A)$中的任意状态 p，通过对 A 的状态转换图进行宽度优先搜索可以找到长度不超过 $|Co(A)|$的输入字 $w\in\Sigma^*$ 使得 $pw\in Nu(A)$。据此可以得到以下结论并设计相应算法。

引理 1[21] 如果自动机 A 的余核中有 m 个状态，则 A 至少有一个长度不超过 m^2 的核字，且该核字可以在 $O(|A|^2)$时间内构造出来。

定义自动机 $A=(Q,\Sigma,\delta)$的对自动机为自动机 $A^2=(Q^2,\Sigma,\delta^2)$，其中：

$$Q^2=\{T:T\subseteq Q,1\leqslant|T|\leqslant 2\}$$

$$\delta^2(T,a)=Ta(T\in Q^2,a\in\Sigma)$$

对于 A 的两个不同状态 p 和 q，如果存在 A 的输入字 w 使得 $pw=qw$，就称$\{p,q\}$是一个同步(状态)对，并称 w 为 p 和 q 的一个同步字；否则，称$\{p,q\}$是一个非同步(状态)对。令 Q'为 A 的所有非同步状态对组成的集合。容易验证 Q 和 Q'都是 A 的子自动机且

$$Nu(A^2)\leqslant Q\cup Q'\leqslant A^2$$

自动机 A^2模其子自动机 $Q\cup Q'$的商自动机 $A^2/(Q\cup Q')$称为 A 的商对自动机，记作 A_q。引理 2 说明 A 的终结字的查找问题等价于 A_q 的核字的查找问题。

引理 2[21] 自动机 A 的一个输入字是终结字当且仅当它是 A_q 的一个核字。

显然，A_q 的核 $Nu(A_q)$仅含一个状态，且该状态在任何输入字母的作用下都不可能转换为其它状态。结合引理 1 和它相应的算法以及引理 2 可以得到如下结论。

定理 1 对于任意自动机 A，算法 1 可以在 $O(|A|^4)$时间内找到 A 的一个终结字。

算法 1 Rystsov 算法

Input: an automaton A=(Q,Σ,δ)

Output: a terminal word of A

1. compute $A_q(A)$
2. S=Co($A_q(A)$)
3. w=ε
4. while exist a state p∈S do
5. find a word v such that pv∈Nu($A_q(A)$)
6. w=wv
7. S=Sv\Nu($A_q(A)$)
8. end while
9. return w

下面用一个实例说明 Rystsov 算法的执行过程，如表 1 所列。

表 1 自动机 A

Table 1 Automaton A

	a	b
0	0	1
1	1	2
2	0	4
3	3	1
4	4	3

首先，计算 A 的商对自动机 A_q，令 nu 为 A_q 核中的唯一状态，如表 2 所列。

表 2 A 的商对自动机 A_q

Table 2 Quotient pair automaton A_q of A

	a	b
nu	nu	nu
{0,2}	nu	{1,4}
{0,3}	{0,3}	nu
{1,4}	{1,4}	{2,3}
{2,3}	{0,3}	{1,4}

对 A_q 的状态转换图进行宽度优先搜索可以找到 A_q 中每个状态的核字，如表 3 所列。

表 3 A_q 状态核字

Table 3 Nuclear words of state A_q

状态	nu	{0,2}	{0,3}	{1,4}	{2,3}
核字	ε	a	b	bab	ab

令 $S=\{\{0,2\},\{0,3\},\{1,4\},\{2,3\}\}$，$w=\varepsilon$。

S 中存在状态$\{0,2\},\{0,3\},\{1,4\},\{2,3\}$，令 $p=\{0,2\}$，令 v 为挑选的状态 p 的核字 a，$w=wv=a$，$S=Sa\backslash Nu(A_q(A))=\{\{0,3\},\{1,4\}\}$。

S 中存在状态$\{0,3\},\{1,4\}$，令 $p=\{0,3\}$，v 为$\{0,3\}$的核字 b，$w=ab$，$S=\{\{2,3\}\}$。

S 中存在状态$\{2,3\}$，令 $p=\{2,3\}$，v 为$\{2,3\}$的核字 ab，$w=abab$，S 为空集，循环结束。此时，w 为 A_q 的一个核字，即 A 的一个终结字，返回 w，算法结束。

3 几个新的终结字查找算法

本节将介绍几个新的终结字查找算法，基于文献[8]的算法 1 本文提出了算法 2 和算法 3，基于文献[15]的算法 3 本文提出了算法 4 和算法 5。本节给出了算法 2 的正确性证明及复杂度分析，由于算法 3－算法 5 都是算法 2 的改进，因此算法 3－算法 5 的正确性证明不再赘述。

首先介绍 Eppstein 预处理[8]。执行新算法之前先执行 Eppstein 预处理，能够有效降低新算法的复杂度。给定一个自动机 $A=(Q,\Sigma,\delta)$，首先构造出 A 的对自动机 A^2(耗时 $O(kn^2)$)，然后对 A^2执行一次宽度优先搜索(耗时 $O(kn^2)$)，搜索结果表示为最短路径森林，森林中的路径将每个同步状态对作用到单个状态上；然后对每个同步状态对$\{p,q\}$以及 A 的每个状态 s，计算 $sw_{\{p,q\}}$，其中 $w_{\{p,q\}}$ 表示同步对$\{p,q\}$的一

个最短同步字，该步骤通过执行一次最短路径森林上的先序遍历来完成(耗时 $O(n^3)$)。因此 Eppstein 预处理的时间复杂度为 $O(kn^2+n^3)$。

对表 1 的自动机 A 进行 Eppstein 预处理可以得到 A 中所有的同步对以及它们的一个同步字(见表 4)，对于每个同步对，也可以得到每个状态在该同步对的同步字作用下的像(见表 5)。

表 4 A 的同步对及其同步字

Table 4 Synchronizing pairs of A and their synchronizing words

同步对	同步字
{0,2}	a
{0,3}	b
{1,4}	bab
{2,3}	ab

表 5 状态的像

Table 5 Images of states

s	0	1	2	3	4
$sw_{\{0,2\}}$	0	1	0	3	4
$sw_{\{0,3\}}$	1	2	4	1	3
$sw_{\{1,4\}}$	2	1	3	2	1
$sw_{\{2,3\}}$	1	2	1	1	3

定理 2 对于任意 n 个状态 k 个字母的自动机 A，算法 2 可以在 $O(kn^2+n^3)$时间内找到 A 的一个终结字。

证明：算法 2 是从当前状态集 S 中挑选一个同步状态对 $\{p,q\}$，将它的同步字 $w_{\{p,q\}}$ 作用在 S 上，直至 S 中的状态互不同步。算法 2 的输出是由所挑选出的诸同步状态对 $\{p,q\}$ 的同步字 $w_{\{p,q\}}$ 顺序连接得到的输入字 w，且算法终止时 $S=Qw$。如果 w 不是 A 的终结字，则：

$$|Qw|>r(A)$$

任取 A 的终结字 w_t，总有：

$$Qww_t=(Qw)w_t\subseteq Qw_t$$

这说明：

$$|Qww_t|\leqslant|Qw_t|$$

因此：

$$|Qw|>r(A)=|Qw_t|=|Qww_t|$$

从而 Qw 中还有同步对。这与 w 的构造方法相矛盾，因此 w 一定是 A 的一个终结字。

算法 2 基本算法

```
Input: an automaton A=(Q,Σ,δ)
Output: a terminal word of A
1. Perform Eppstein Pre-processing
2.   S=Q
3.   w=ε
4.   while exist a synchronizing pair {p,q} in S do
5.   S=Sw{p,q}
6.   w=ww{p,q}
7. end while
8. return w
```

算法 2 的第 1 行 Eppstein 预处理耗时 $O(kn^2+n^3)$。在内层循环中，第 4 行从 S 中挑选一个同步状态对 $\{p,q\}$ 耗时 $O(n^2)$，第 5 行在 Eppstein 预处理的基础上计算 $Sw_{\{p,q\}}$ 耗时 $O(n)$，因此执行一次内层循环耗时 $O(n^2)$。因为每执行一次内层循环 S 的基数至少减 1，所以内层循环最多执行 $n-1$ 次，因而算法 2 的时间复杂度为 $O(kn^2+n^3)$。证毕。

下面用表 1 的自动机 A 说明算法 2 的执行过程。

对 A 执行 Eppstein 预处理得表 4 与表 5；

令 $S=Q=\{0,1,2,3,4\}$，$w=\varepsilon$，进入主循环；

S 中存在同步对{0,2}，{0,3}，{1,4}，{2,3}，挑选一个同步对{0,2}，$S=Sw_{\{0,2\}}=\{0,1,3,4\}$，$w=ww_{\{0,2\}}=a$；

S 中存在同步对{0,3}与{1,4}，挑选一个同步对{0,3}，$S=\{1,2,3\}$，$w=ab$；

S 中存在同步对{2,3}，挑选同步对{2,3}，$S=\{1,2\}$，$w=abab$，S 中不再存在同步对，循环结束，此时 w 即为 A 的一个终结字，返回 w，算法结束。

算法 3

```
Input: an automaton A=(Q,Σ,δ)
Output: a terminal word of A
1. Perform Eppstein Pre-processing
2.   S=Q
3.   w=ε
4.   while exist a synchronizing pair in S do
5.   find a synchronizing pair {p,q} in S such that the word w{p,q} is
     shortest
6.   S=Sw{p,q}
7.   w=ww{p,q}
8. end while
9. return w
```

在算法 2 的基础上，在每次内层循环中，算法 3 总从 S 中选取最短同步字长度最小的同步对 $\{p,q\}$，这样并不会比算法 2 耗费更多时间。由此我们可以得出以下推论 1。

推论 1 对于任意 n 个状态 k 个字母的自动机 A，算法 3 可以在 $O(kn^2+n^3)$时间内找到 A 的一个终结字。

下面用表 1 的自动机 A 说明算法 3 的执行过程。

给定自动机 A，执行 Eppstein 预处理得到表 4 与表 5，$S=\{0,1,2,3,4\}$，$w=\varepsilon$，进入主循环；S 中存在同步对{0,2}，{0,3}，{1,4}，{2,3}，由于{0,2}的同步字最短，所以挑选同步对{0,2}，$S=Sw_{\{0,2\}}=\{0,1,3,4\}$，$w=ww_{\{0,2\}}=a$，第一步循环结束，剩余步骤不再赘述。

对自动机 $A=(Q,\Sigma,\delta)$状态集的子集 S，令 $D(S)$为 S 中所有同步对的集合，并定义：

$$\Phi(S)=\sum_{\{p,q\}\in D(S)}|w_{\{p,q\}}|$$

当$\{p,q\}\in D(S)$时，定义：

$$\Phi_{\{p,q\}}(S)=\Phi(Sw_{\{p,q\}})-\Phi(S)$$

$$\Phi'_{\{p,q\}}(S)=\Phi_{\{p,q\}}(S)+|w_{\{p,q\}}|$$

如果在算法 2 的每次内层循环中总从 S 中选取 $\Phi_{\{p,q\}}(S)$ 和 $\Phi'_{\{p,q\}}(S)$最小的同步对 $\{p,q\}$，就又可以得到两种新的终结字查找算法。

在算法 2 的基础上，在每次内层循环中，算法 4 总从 S 中选取 $\Phi_{\{p,q\}}(S)$最小的同步对 $\{p,q\}$，可知对于 S 中任意一个同步对 $\{p,q\}$，计算 $\Phi_{\{p,q\}}(S)$耗时 $O(n^2)$，为了在 S 中选择 $\Phi_{\{p,q\}}(S)$和 $\Phi'_{\{p,q\}}(S)$函数值最小的同步对，最多需要计算 $O(n^2)$个同步对的函数值，此时，在 S 中挑选一个同步对将会耗时 $O(n^4)$，执行一次内层循环将耗时 $O(n^4)$，因为最多执行 $n-1$ 次内层循环，且算法 4 开始时会执行一次 Eppstein 预处理，算法 4 的时间复杂度将会变为 $O(kn^2+n^5)$。由此我们可以得到以下推论 2。

算法 4

```
Input: an automaton A=(Q,Σ,δ)
```

Output: a terminal word of A
1. Perform Eppstein Pre-processing
2. S=Q
3. w=ε
4. while exist a synchronizing pair in S do
5. find a synchronizing pair {p,q} in S such that $\Phi_{\{p,q\}}(S)$ is minimum
6. S=$Sw_{\{p,q\}}$
7. w=$ww_{\{p,q\}}$
8. end while
9. return w

推论 2 对于任意 n 个状态 k 个字母的自动机 A，算法 4 可以在 $O(kn^2+n^5)$时间内找到 A 的一个终结字。

下面用表 1 的自动机 A 说明算法 4 的执行过程。

给定自动机 A，执行 Eppstein 预处理得到表 4 与表 5，$S=\{0,1,2,3,4\}$，$w=\varepsilon$，进入主循环；S 中存在同步对$\{0,2\}$，$\{0,3\}$，$\{1,4\}$，$\{2,3\}$，对于每个同步对$\{p,q\}$，我们计算 $\Phi_{\{p,q\}}$ 的函数值：

$\Phi_{\{0,2\}}=\Phi(Sw_{\{0,2\}})-\Phi(S)=-3$

$\Phi_{\{0,3\}}=-2$，$\Phi_{\{1,4\}}=-5$，$\Phi_{\{2,3\}}=-5$

由于$\{1,4\}$的函数值最小，所以挑选同步对$\{1,4\}$，$S=Sw_{\{1,4\}}=\{1,2,3\}$，$w=ww_{\{1,4\}}=bab$，第一步循环结束，剩余步骤不再赘述。

将算法 4 的 $\Phi_{\{p,q\}}(S)$函数用 $\Phi'_{\{p,q\}}(S)$函数替换，便可以得到算法 5。在内层循环中，无论是使用 $\Phi_{\{p,q\}}(S)$函数还是使用 $\Phi'_{\{p,q\}}(S)$函数挑选同步对，都将耗时 $O(n^2)$，因此，我们可以得到以下推论 3。

算法 5

Input: an automaton A=(Q,Σ,δ)
Output: a terminal word of A
1. Perform Eppstein Pre-processing
2. S=Q
3. w=ε
4. while exist a synchronizing pair in S do
5. find a synchronizing pair {p,q} in S such that $\Phi'_{\{p,q\}}(S)$ is minimum
6. S=$Sw_{\{p,q\}}$
7. w=$ww_{\{p,q\}}$
8. end while
9. return w

推论 3 对于任意 n 个状态 k 个字母的自动机 A，算法 5 可以在 $O(kn^2+n^5)$时间内找到 A 的一个终结字。

对于算法 5 我们不再使用实例进行说明。

4 结果及分析

本文实现了算法 1—算法 5 并进行了相关实验。算法实现语言为 C++，编译器为 g++。实验在 Dell Precision T3610 计算机上进行，其配置如下：处理器：Intel® Xeon® CPU E5-1650 V2 @ 3.50GHz，内存为 32GB，操作系统为 64 位 Linux 操作系统。

实验样本的状态数设定为 $n=20,40,60,\cdots,200$，对每个 n 随机生成 1000 个 n 状态自动机。实验结果如表 6、表 7 和图 1—图 3 所示。其中，表 6 是算法对各种状态数的样本的平均执行时间，表 7 是算法对各种状态数的样本找到的终结字的平均长度，图 1 和图 2 分别是表 6 和表 7 的图形表示。由于在图 1 中无法直观地比较算法 1—算法 3 的平均执行时间，因此在图 3 中对这 3 种算法的平均执行时间单独进行比较。

表 6 平均执行时间

Table 6 Average running time

time(n)/ms	算法 1	算法 2	算法 3	算法 4	算法 5
20	0	0	0	0	0
40	0	0	0	0.011	0.021
60	0	0	0	1.918	1.981
80	0.022	0	0	7.951	7.963
100	0.096	0.002	0.048	17.813	18.062
120	0.506	0.027	0.099	34.219	34.298
140	1.078	0.101	0.176	66.515	59.347
160	1.373	0.191	1.033	93.234	95.165
180	2.045	1.039	1.060	147.91	143.216
200	2.757	1.082	1.145	216.438	210.579
Sum	7.877	2.442	3.561	586.009	570.632

表 7 终结字平均长度

Table 7 Average length of terminal words

length(n)	算法 1	算法 2	算法 3	算法 4	算法 5
20	14.984	14.843	13.586	11.730	11.352
40	24.630	24.685	21.436	18.400	18.014
60	32.599	32.528	27.563	23.482	22.987
80	39.362	39.245	32.603	27.513	26.952
100	45.000	45.180	36.933	30.817	30.288
120	51.190	51.422	41.542	35.137	34.428
140	56.158	56.210	45.239	37.863	37.057
160	61.588	61.870	49.372	41.268	40.548
180	66.529	65.983	52.431	43.732	43.022
200	71.104	70.639	55.568	46.420	45.510
Sum	463.144	462.605	376.273	316.362	310.158

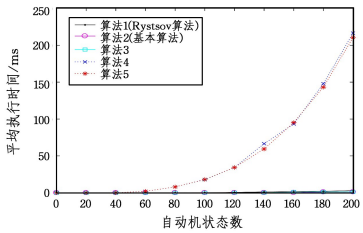

图 1 算法平均执行时间

Fig. 1 Average running time

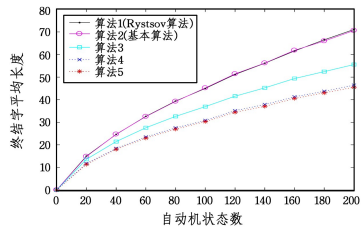

图 2 算法查找终结字平均长度

Fig. 2 Average length of terminal words

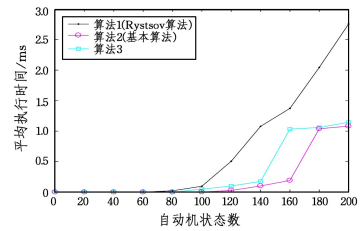

图 3 算法 1－算法 3 的平均执行时间

Fig. 3 Average running time of algorithm 1－3

实验结果表明：在平均执行时间方面，有：

算法 2≤算法 3≤Rystsov 算法≤算法 4≈算法 5

在查找到的自动机终结字平均长度方面，有：

算法 5≤算法 4≤算法 3≤算法 2≈Rystsov 算法

由实验结果可以发现，Rystsov 算法无论是在执行效率方面还是计算结果方面都没有优势，因此算法 2－算法 5 都可以认为是 Rystsov 算法的改进。

参 考 文 献

[1] GROSSMAN D D, BLASGEN M W. Orienting mechanical parts by computer-controlled manipulator [J]. IEEE Transactions on Systems, Man, and Cybernetics, 1975(5): 561-565.

[2] NATARAJAN B K. An algorithmic approach to the automated design of parts orienters [C] // 27th Annual Symposium on Foundations of Computer Science. IEEE, 1986: 132-146.

[3] ERDMANN M A, MASON M T. An exploration of sensorless manipulation[J]. IEEE Journal on Robotics and Automation, 1988, 4(4): 369-379.

[4] NATARAJAN B K. Some paradigms for the automated design of parts feeders[J]. The International journal of robotics research, 1989, 8(6): 98-109.

[5] ČERNÝ J. Poznamka k homogenym eksperimentom s konechnymi automatami, Math[J]. Fyz. Cas, 1964, 14: 208-215.

[6] STARKE P H. Eine Bemerkung über homogene Experimente [J]. Elektronische Informationsverarbeitung und Kybernetik, 1966, 2(4): 257-259.

[7] ČERNÝ J, PIRICKÁ A, ROSENAUEROVÁ B. On directable automata[J]. Kybernetika, 1971, 7(4): 289-298.

[8] EPPSTEIN D. Reset sequences for monotonic automata[J]. SIAM Journal on Computing, 1990, 19(3): 500-510.

[9] DUBUC L. Sur les automates circulaires et la conjecture de černy [J]. RAIRO-Theoretical Informatics and Applications, 1998, 32(1/2/3): 21-34.

[10] KARI J. Synchronizing finite automata on Eulerian digraphs[J]. Theoretical Computer Science, 2003, 295(1/2/3): 223-232.

[11] ANANICHEV D S, VOLKOV M V. Synchronizing monotonic automata[J]. Theoretical Computer Science, 2004, 327(3): 225-239.

[12] ANANICHEV D S. The mortality threshold for partially monotonic automata[C] // International Conference on Developments in Language Theory. Springer, Berlin, 2005: 112-121.

[13] ANANICHEV D S, VOLKOV M V. Synchronizing generalized monotonic automata [J]. Theoretical Computer Science, 2005, 330(1): 3-13.

[14] VOLKOV M V. Synchronizing automata preserving a chain of partial orders[J]. Theoretical Computer Science, 2009, 410(37): 3513-3519.

[15] ROMAN A. Synchronizing finite automata with short reset words [J]. Applied Mathematics and Computation, 2009, 209(1): 125-136.

[16] ANANICHEV D, GUSEV V, VOLKOV M. Slowly synchronizing automata and digraphs[C] // International Symposium on Mathematical Foundations of Computer Science. 2010: 55-65.

[17] STEINBERG B. The Černý conjecture for one-cluster automata with prime length cycle [J]. Theoretical Computer Science, 2011, 412(39): 5487-5491.

[18] BERLINKOV M V. Synchronizing automata on quasi-eulerian digraph[C] // International Conference on Implementation and Application of Automata. Springer, Berlin, 2012: 90-100.

[19] CHEN X P, HE Y, XIAO F F. Synchronization of Certain Family of Automata and Consumption Function Analysis[J]. Computer Science, 2019, 46(S2): 535-538.

[20] CUI Z H, HE Y, SUN S Y. Synchronizing Bounded Partially Ordered Automata[J]. Chinese Journal of Computers, 2019, 42: 610-623.

[21] RYSTSOV I K. Rank of a finite automaton[J]. Cybernetics and Systems Analysis, 1992, 28(3): 323-328.

[22] ALMEIDA J, STEINBERG B. Matrix mortality and the Černý-Pin conjecture[C] // International Conference on Developments in Language Theory. Springer, Berlin, Heidelberg, 2009: 67-80.

SUN Shi-yuan, born in 1995, master, is a member of China Computer Federation. His main research interest include theory of automaton and so on.

HE Yong, born in 1970, Ph. D, professor, is a member of China Computer Federation. His main research interests include Formal language and automata theory.