ISBN 978-952-5726-06-0
Proceedings of the 2009 International Workshop on Information Security and Application (IWISA 2009)
Qingdao, China, November 21-22, 2009

An Algorithm for Minimizing a Completely Deterministic Finite Automaton Based on State Transition Inverse Mapping

Chu Chen[1], Pinghong Ren[1], Jiguo Yu[1], Nong Yu[2]
[1] College of Computer Science, Qufu Normal University, Rizhao, China
Email: {chenchu, research, jiguoyu}@yeah.net
[2] College of Information and Engineering, Shandong University of Science and Technology, Qingdao, China
Email: yunong@sdkd.net.cn

***Abstract*—During the process of minimizing a deterministic finite automaton, there exist computing dependency and repetitive computing problems caused by the disorder of selecting a set to partition in the partition algorithm or by the disorder of computing state-pairs in the combination algorithm. To solve theses problems, a new algorithm for minimizing a completely deterministic finite automaton is presented. Firstly, based on the concept of a completely deterministic finite automaton, characteristics of state transition inverse mapping and the usage to reduce repetitive computing are analyzed. Secondly, the algorithm for minimizing a completely deterministic finite automaton, its rightness proof and analysis of time complexity are given. In the end, an experiment is carried out and the result shows that this algorithm is fairly efficient for minimizing a finite automaton.**

***Index Terms*—minimizing, completely deterministic, finite automaton, state transition, inverse mapping**

I. INTRODUCTION

As the basis of computer science, finite automaton (FA) theory can be applied to the abstraction of most finite systems in computer field. Deterministic finite automaton (DFA) is a very important part of FA theory and its minimization has important theoretical and practical value. There mainly exist two kinds of DFA minimization algorithms which are both based on analysis of states: the partition algorithm[1][2][3] and the combination algorithm[4]. However, computing dependency and repetitive computing problems may be brought by the disorder of selecting partition sets in the partition algorithm or by the disorder of computing state-pairs[4] in the combination algorithm. To solve these problems, based on FA theory[5][6] and analysis of characteristics of completely deterministic finite automaton, a new algorithm is developed, meanwhile, its rightness proof, time complexity and efficiency are presented.

Foundation: the Award for Excellent Young Scientists Foundation of Shandong Province of China under Grant No.2005BS01016; the Initial Scientific Research Foundation of Qufu Normal University under Grant No.2004032.
Author introduction: Chu Chen, lecturer, research field: intelligent information processing; Pinghong Ren, lecturer, research fields: CG and algorithm; Jiguo Yu, professor, research field: algorithm; Nong Yu, professor, research field: real-time algorithm.

II. DEFINITION, TRANSITION AND CHARACTERISTIC ANALYSIS OF COMPLETELY DETERMINISTIC FA

Definition 1 DFA $M=(Q,\Sigma,\delta,q_0,F)$ is a completely deterministic finite automaton(CDFA)$\Leftrightarrow\forall q\in Q$, $\forall a\in\Sigma$, $\delta(q,a)\in Q$. If $|\Sigma|=0$ and $|Q|>0$ ($\delta=\varnothing$, $RL_M=\varnothing$ or $\{\varepsilon\}$), M is a CDFA. If $|Q|=0$, it's no use to discuss.

Definition 2[1] $\exists$DFA $M=(Q,\Sigma,\delta,q_0,F)$, if $\exists q\in Q$, $a\in\Sigma$, $\delta(q,a)\notin\delta$, to generate a state $q_{dead}\notin Q(q_{dead}\notin F)$ and to modify δ: $\forall q\in Q$, $a\in\Sigma$, if $\delta(q,a)\notin\delta$, $\delta(q,a)=q_{dead}$, $\forall a\in\Sigma$, $\delta(q_{dead},a)=q_{dead}$. q_{dead} is called a dead state.

Property 1 $\exists$CDFA $M=(Q,\Sigma,\delta,q_0,F)$, if $\exists q_{dead}\in Q$, $\forall q\in Q-\{q_{dead}\}$, $q_{dead}\exists q$.

Proof. $\forall a\in\Sigma$, $(\delta(q_{dead},a)=q_{dead})\wedge(q_{dead}\notin F)$, $\forall\alpha\in\Sigma^*$, $\delta(q_{dead},\alpha)\in F$. Hence, $\forall q\in Q-\{q_{dead}\}$, $q_{dead}\exists q$. □

Theorem 1 $\exists$DFA M, $\exists$CDFA M', $M\equiv M'$.

Proof.

(1) If DFA M is a CDFA, $M=M'$, obviously $M\equiv M'$.

(2) If DFA M is not a CDFA, a CDFA with only one dead state can be constructed: $M'=(Q\cup\{q_{dead}\},\Sigma,\delta',q_0,F)$, $q_{dead}\notin Q$, $\delta'=\delta\cup\{\delta(q,a)=q_{dead}|\exists q\in Q,\ a\in\Sigma,\ \text{if}\ \delta(q,a)\notin Q\}\cup\{\delta(q_{dead},a)=q_{dead}|\forall a\in\Sigma\}$.

(2.a) $\forall\alpha\in L(M)$, $(\delta\exists\delta')\forall\alpha\in L(M')$;

(2.b) $\forall\alpha\in L(M')$, $(\delta'-\delta=(\{\delta(q,a)=q_{dead}|\exists q\in Q,\ a\in\Sigma,\ \text{if}\ \delta(q,a)\notin Q\}\cup\{\delta(q_{dead},a)=q_{dead}|\forall a\in\Sigma\}))\wedge(q_{dead}\notin F)\exists(\exists\alpha_1,\alpha_2,\ (\alpha_1\cdot\alpha_2=\alpha)\wedge(\delta(q_0,\alpha_1)=q_{dead})\wedge(\delta(q_{dead},\alpha_2)\in F))$, hence, $\alpha\in L(M)$.

According to (2.a) and (2.b), it's obvious that $M\equiv M'$.

According to (1) and (2), the theorem is right. □

Theorem 2 $\exists$DFA M,DFA M',CDFA M'',$(((M\equiv M'')\wedge(M'$ is minimization of $M))\exists$(in the sense of isomorphism, M' is minimization of $M''))\wedge(((M\equiv M'')\wedge(M'$ is minimization of $M''))\exists$(in the sense of isomorphism, M' is minimization of M)).

Proof. $(M\equiv M'')\exists((L(M)=L(M''))\wedge(L(M)\exists RL)\wedge(L(M'')\exists RL))$, it can be inferred that there is only one minimal DFA of M and M'':$((M'$ is minimization of $M)\exists$(in the sense of isomorphism, M' is minimization of $M''))\wedge((M'$ is minimization of $M'')\exists$(in the sense of isomorphism, M' is minimization of M)). □

Corollary 1 $\exists$DFA M, if M is not a CDFA, a CDFA M' equivalent to M can be constructed according to

Definition 2. The dead state and corresponding state transition functions introduced in the construction do not affect the equivalence and uniqueness of the minimization result of M' and M.

Proof. It can be inferred by Theorem 2 and the uniqueness of minimal DFA in the sense of isomorphism. □

Corollary 2 During the minimization of CDFA, dead states if exist may not be computed in judging equivalent states.

Proof. It can be proved by using Property 1 and Corollary 1. □

Theorem 3 □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, □$q\in Q$, $a\in\Sigma$, $\delta^{-1}(q,a)=\{q'|q'\in Q\wedge\delta(q',a)=q\}$:

(1) □q, $q'\in Q$, $a\in\Sigma$, $q\neq q'$, $\delta^{-1}(q,a)\cap\delta^{-1}(q',a)=\varnothing$.

(2) $(\forall a\in\Sigma)\bigcup_{\forall q\in Q}\delta^{-1}(q,a)=Q$.

Proof. (1) Reduction to absurdity: □q, $q'\in Q$, $a\in\Sigma$, $q\neq q'$, $\delta^{-1}(q,a)\cap\delta^{-1}(q',a)\neq\varnothing$. □$q\in Q$, $a\in\Sigma$, $\delta^{-1}(q,a)=\{q''|q''\in Q\wedge\delta(q'',a)=q\}$, □$q'\in Q$, $a\in\Sigma$, $\delta^{-1}(q',a)=\{q'''|q'''\in Q\wedge\delta(q''',a)=q\}$, $\delta^{-1}(q,a)\cap\delta^{-1}(q',a)=\{p|p\in Q\wedge\delta(p,a)=q\wedge\delta(p,a)=q'\}\neq\varnothing$. It can be known by Definition 1: $q=q'$, and the result is absurd considering the condition $q\neq q'$. Hence, □q, $q'\in Q$, $a\in\Sigma$, $q\neq q'$, $\delta^{-1}(q,a)\cap\delta^{-1}(q',a)=\varnothing$.

(2) CDFA $M=(Q,\Sigma,\delta,q_0,F)$, □$q'\in Q$, $a\in\Sigma$, $\delta(q',a)=q\in Q$, hence, $(\forall a\in\Sigma)\bigcup_{\forall q\in Q}\delta^{-1}(q,a)=Q$. □

Corollary 3 If P□Q, the following conclusion is right: $(\forall a\in\Sigma)(\bigcup_{\forall q\in P}\delta^{-1}(q,a)\cap\bigcup_{\forall q\in Q-P}\delta^{-1}(q,a))=\phi$.

Proof. It can be proved by using Theorem 3(1). □

Corollary 4 If P□Q, the following conclusion is right: $(\forall a\in\Sigma)(\bigcup_{\forall q\in P}\delta^{-1}(q,a)\cup\bigcup_{\forall q\in Q-P}\delta^{-1}(q,a))=Q$.

Proof. It can be proved by using Theorem 3(2). □

Definition 3 □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, let R_E is the equivalent relation on M and R_N is the nonequivalent relation on M. □W□Q, V□Q, $W\divideontimes V=\{(a,b)|((□a\in W\wedge□b\in V)\vee(□a\in V\wedge□b\in W))\wedge((aR_Eb\vee aR_Nb)□((a,b)=(b,a)))\wedge(a\neq b)\}$.

Definition 4 Non-effective computing states set for 'a'(NECSS[a]): □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, NECSS[$a$]=$\{q|a\in\Sigma,□q\in Q,\delta^{-1}(q,a)=\varnothing\}$.

Definition 5 Full effective computing states set for 'a'(FECSS[a]): □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, FECSS[$a$]=$\{q|a\in\Sigma,□q\in Q,\delta^{-1}(q,a)\neq\varnothing\}$.

Property 2 □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, (NECSS[$a$]∩FECSS[$a$]=ø)∧(NECSS[$a$]∪FECSS[$a$]=$Q$)

Proof. It can be proved by Definition 4, Definition 5 and Theorem 3. □

Theorem 4 □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, □$a\in\Sigma$, □$X$□$Q$, $X'=\{q|□p\in X,\delta^{-1}(p,a)=q\}$:

(1) (X□NECSS[a])□(X'=ø)∧(Q-X'=Q), (FECSS[a]□X)□(X'=Q)∧(Q-X'=ø).

(2) (X□NECSS[a]∨FECSS[a]□X)□(□b∈X', $c\in Q$-X', b□c).

Proof. (1.a) (X□NECSS[a])□(□$q\in X$□NECSS[a],$\delta^{-1}(q,a)$=ø)□(X'=ø)□(Q-X'=Q)□(X'=ø)∧(Q-X'=Q).

(1.b) (FECSS[a]□X)□(X'=$X'_{FECSS[a]}$+$X'_{X\text{-}FECSS[a]}$=$\{q|□p\in$ FECSS[a],$\delta^{-1}(p,a)=q\}+\{q'|□p'\in(X\text{-FECSS}[a])□(Q\text{-}$FECSS[$a$]),$\delta^{-1}(p',a)=q'\}$)□($X'$=$X'_{FECSS[a]}$+$X'_{X\text{-}FECSS[a]}$=$\{q|□p$ $\in$FECSS[a],$\delta^{-1}(p,a)=q\}+\{q'|□p'\in(X\text{-FECSS}[a])□$NECSS [$a$],$\delta^{-1}(p',a)=q'\}$)□($X'$=$X'_{FECSS[a]}$+$X'_{X\text{-}FECSS[a]}$=$\{q|□p\in$FECSS [$a$],$\delta^{-1}(p,a)=q\}+$ø)□($X'$=$X'_{FECSS[a]}$=$\{q|□p\in$FECSS[$a$],$\delta^{-1}(p,a)$ =$q\}$)□(X'=$X'_{FECSS[a]}$=Q)□(Q-X'=ø)□ ((X'=Q)∧(Q-X'=ø)).

Theorem 4(1) has been proved by (1.a) and (1.b).

(2.a) (X□NECSS[a])□((X'=ø)∧(Q-X'=Q))□(□$b\in X'$, $c\in Q$-X',b□c);

(2.b) (FECSS[a]□X)□((X'=Q)∧(Q-X'=ø))□(□$b\in X'$, $c\in Q$-X',b□c).

Theorem 4(2) has been proved by (2.a) and (2.b).

Theorem 4 is right. □

Definition 6 A set X is the effective computing kernel for 'a' (ECK[X,a]): □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, □$a\in\Sigma$, □$X$□ Q, ECK[X,a]=X-NECSS[a].

Definition 7 A set X is the effective computing kernel item for 'a' (ECKI[X,a]): □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, □$a\in\Sigma$, □$X$□$Q$, ECKI[$X$,$a$]=<ECK[$X$,$a$],$a$>.

Theorem 5 □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, □$a\in\Sigma$, □$X$□$Q$, δ^{-1}(ECK[X,a],a)=$\delta^{-1}(X,a)$.

Proof. δ^{-1}(ECK[X,a],a)=δ^{-1}(X-NECSS[a],a)=$\{q|□p\in X\wedge p\notin$NECSS[$a$], $\delta^{-1}(p,a)=q\}=\{q'|□p'\in X$, $\delta^{-1}(p',a)=q'\}$-$\{q''|□p''\in X\wedge p''\in$NECSS[$a$], $\delta^{-1}(p'',a)=q''\}=\{q'|□p'\in X$, $\delta^{-1}(p',a)=q'\}$-ø =$\{q'|□p'\in X$, $\delta^{-1}(p',a)=q'\}=\delta^{-1}(X,a)$.

Definition 8 □CDFA $M=(Q,\Sigma,\delta,q_0,F)$, nonequivalent state-pairs set (NESPS)=$\{(a,b)|a\in Q\wedge b\in Q\wedge a\neq b\wedge a□b\}$, nonequivalent states set-pairs set (NESSPS)=$\{<X,Y>|X\subseteq Q\wedge Y\subseteq Q\wedge X□Y\}$. If $<X,Q\text{-}X>\notin$NESSPS∧(□$(a,b)\in(X\divideontimes(Q\text{-}X))\wedge(a,b)\notin$NESPS ∧ a□b), $<X,Q\text{-}X>$ is called effective state set-pairs, otherwise it is called non-equivalent state set-pairs.

III. CDFA Minimization Algorithm Based on State Transition Inverse Mapping

Based on the concept and characteristic analysis of CDFA in the second part, core of broad-first CDFA minimization algorithm based on state transition inverse mapping is given in pseudo-code as follows.

Input: meaningful CDFA $M=(Q,\Sigma,\delta,q_0,F)$.

Output: if M can be minimized, the result is the minimal DFA.

Method:

1) if (F==ø){printf("No equivalent states in M"); goto 9);}
2) if (|Σ|==0) goto 9);
3) $NESS=(Q\text{-}F)\divideontimes F$, $SSW=(Q\text{-}F)\divideontimes(Q\text{-}F)+F\divideontimes F$. /*$NESS$ is a set of nonequivalent state-pairs and SSW is a set of state-pairs waiting to be judged*/
4) if (SSW==ø){printf("No equivalent states in M"); goto 9);}
5) Compute δ^{-1}, NECSS[a] and FECSS[a] for every $a\in\Sigma$.
6) if(|F|≤|Q-F|){$min=F$,$max=(Q\text{-}F)$,$D=\{<min,max>\}$,$W=$ø,$T=$ø} else {$min=(Q\text{-}F)$,$max=F$,$D=\{<min,max>\}$,$W=$ø,$T=$ø}.

/*D is a set of state-set-pairs being computed, W is a set of state-set-pairs waiting to be computed, T is a set of computed ECKI[X,y]*/

7) while(((*D*≠ø)||(*W*≠ø))&&(*SSW*≠ø)){
   To fetch a member of *D* and let *current*=<*min*,*max*>;
   for (every $a\in\Sigma$){
    if (!(*min*□NECSS[*a*]∨FECSS[*a*]□*min*)){
     ECK[*min*,*a*]=*min*-NECSS[*a*];
     if (ECKI[*min*,*a*]∉*T*){
      ECK[*max*,*a*]=*max*-NECSS[*a*];
      newmin={*p*|*p*∈δ^{-1}(*q*,*a*),□*q*∈ECK[*min*, *a*]};
      newmax=*Q*-*newmin*;
      if (|*newmin*|>|*newmax*|) *newmin*↔*newmax*;
      W+={<*newmin*,*newmax*>};
      T+={<ECK[*min*,*a*],*a*>,<ECK[*max*,*a*],*a*>};
      SSW-=*newmin*※*newmax*;
      if (*SSW*==ø) exit this 'for' cycle;
     }//end of 'if (ECK[*min*,*a*]∉*T*)'
    }// 'if !(*min*□NECSS[*a*]∨FECSS[*a*]□*min*)'
    D-=*current*;
    if ((*D*==ø)&&(*W*≠ø)) *D*↔*W*;
   }//end of 'for (every $a\in\Sigma$)'
  }//end of 'while'

8) To combine equivalent states same as the way adopted by most DFA minimization algorithms and to deal with Q,δ,q_0 and F.

9) Q'=Q,δ'=δ,q_0'=q_0,F'=F. The minimal DFA is M'=(Q',Σ,δ',q_0',F').

IV. Rightness Proof

Theorem 6 The algorithm can recognize all non-equivalent state-pairs and has no mistake.

Proof. First, it is to be proved that the algorithm has no mistake in judging nonequivalent state-pairs.

The algorithm produces two $|\Sigma|$-branch trees whose roots are F and Q-F separately. Corresponding nodes in the trees are nonequivalent.

(1) Let k is the depth of the tree. k=0: every state-pairs of F※(Q-F) are nonequivalent according to Property 1.

(2) Assumption: when k=m (m is less than the max depth of the tree) the algorithm has no mistake in judging nonequivalent state-pairs. It needs to be proved that when k=m+1 the conclusion is still right. Choose freely a node at the layer of m+1 in the tree whose root is F and note it as S_1'. S_1' must come from S_1 at the layer of m in the same tree through the computing δ^{-1}(ECK [S_1,x],x). Correspondingly, the node S_2' at the layer of m+1 in the tree whose root is Q-F, must come from S_2 at the layer of m in the tree whose root is Q-F through the computing δ^{-1}(ECK[S_2,x],x). It can be inferred from the assumption that state-pairs of S_1※S_2 are nonequivalent. □$q_1\in S_1'$, □$q_2\in S_2'$,$\delta(q_1,x)\in S_1$,$\delta(q_2,x)\in S_2$,q_1□q_2. Similar proof can be made if choosing freely a node at the layer of m+1 in the tree whose root is Q-F. Hence, when k=m+1 the algorithm has no mistake in judging nonequivalent state-pairs.

It can be concluded from the induction that the algorithm has no mistake in judging nonequivalent state-pairs.

Second, it is to be proved that the algorithm can recognize all nonequivalent state-pairs without omission.

The algorithm computes δ^{-1}(ECK[S,y],y)(□$y\in\Sigma$) for every node S in the trees whose root are F and Q-F. And it is equal to computing δ^{-1}(S,y)(□$y\in\Sigma$) according to Theorem 5. There are two conditions to end the branch computing: one is ø according to Theorem 4 and it's no use to continue computing; the other described in Definition 8 can lead to repetitive computing and it is no use to continue too. So the algorithm can compute all useful branches without omission. □

V. Time Complexity Analysis

Theorem 7 There are at least two new nonequivalent states between corresponding state sets S' and Q-S' in the $|\Sigma|$-branch trees whose roots are F and Q-F.

Proof. To choose freely two state sets S' and S''($S'\neq S''$) in the tree whose root is F and there exist Q-S' and Q-S'' in the $|\Sigma|$-branch tree whose root is Q-F' corresponding to S' and S''. It can be inferred by Theorem 6 that:(Q-S')≠(Q-S'')∧(Q-S')≠S''∧(Q-S'')≠S'∧S'≠ø∧S''≠ø∧(Q-S')≠ø∧(Q-S'')≠ø. There exists one state-pair (x,y) and (($x\in S'\wedge y\in$ (Q-S'))∨($y\in S'\wedge x\in$(Q-S')))∧(($x\in S''\wedge y\in S''$)∨($x\in$(Q-S'')∧$y\in$(Q-S'')). It also can be inferred by Theorem 6 that x is not equivalent to y. According to the free choice of S'', (x,y) is a new nonequivalent state-pair. □

Corollary 5 There are at least two new nonequivalent states produced by S' (different from F and Q-F) and Q-S' but not by F or Q-F.

Proof. It can be proved by using reduction to absurdity method according to Theorem 7. □

Theorem 8 The algorithm's worst time complexity is $(\frac{|Q|\cdot|Q-1|}{2}-|Q-F|\cdot|F|)\cdot\left\lfloor\frac{|Q|}{2}\right\rfloor+|Q|\cdot|\Sigma|$.

Proof. The time complexity of the fifth step is $|Q|\cdot|\Sigma|$. To assume that there exist k nonequivalent state-pairs, the number of nonequivalent state-pairs which need to be recognized is $(k-|Q-F|\cdot|F|)$ according to Corollary 5. Every time the algorithm selects one of S' and Q-S' that includes less states to compute, so the max number states to compute every time is $\left\lfloor\frac{|Q|}{2}\right\rfloor$. When it come to the worst case that is k=$|Q|\cdot|Q-1|$, the worst time complexity to recognize nonequivalent state-pairs is $(\frac{|Q|\cdot|Q-1|}{2}-|Q-F|\cdot|F|)\cdot\left\lfloor\frac{|Q|}{2}\right\rfloor+|Q|\cdot|\Sigma|$. □

VI. Advantages of the Algorithm and Its Efficiency

The algorithm can deal with all kinds of situations. Not only CDFA can be minimized, but also all DFA such as DFA without dead states, DFA with more than one dead state and more complex DFA mixed by the two kinds of DFAs. The algorithm use structural characteristics of

CDFA to simplify computing and has no problem[4] in the partition algorithm or in the combination algorithm.

For example[4]: DFA M=({s,a,b},{0,1},{δ(s,0)=a, δ(s, 1)=b, δ(b,1)=a},s,{a,b}). Above all, DFA M needs to be transformed to its equivalent CDFA :M'=({s,a,b,q},{0,1}, {δ(s,0)=a,δ(s,1)=b,δ(b,1)=a,δ(b,0)=q,δ(a,0)=q,δ(a,1)=q,δ (q,0)=q,δ(q,1)=q},s,{a,b}). After the transformation, M' is minimized according to the steps of the algorithm. For the first step and the second step, F={a,b}≠ø and $|\Sigma|$=2≠0. For the third step: $NESS$={(s,a), (s,b), (q,a), (q,b)}, SSW={(s,q), (a,b)}. SSW≠ø so for the fifth step: δ^{-1}={δ^{-1} (a,0)={s},δ^{-1}(a,1)={b},δ^{-1}(b,1)={s},δ^{-1}(q,0)={a,b,q},δ^{-1}(q,1) ={a,q}},NECSS[0]={s,b},NECSS[1]={s},FECSS[0]={a, q},FECSS[1]={a,b,q}. For the sixth step: |{a,b}|=|{s,q}| and it's feasible to let min=F or min=Q-F. Here Let min=F and max= Q-F. D=<{a,b},{s,q}>,W=ø,T=ø. For the seventh step: D≠ø and SSW≠ø so the cycle can be executed. For 0∈Σ, (min□NECSS[0] ∨FECSS[0]□min) is false, therefore, ECK [min,0]=min-NECSS[0]={a,b}-{s,b}={a}. For ECKI[min,0]∉T, $newmin$={s}, $newmax$= {s,a,b,q}-{s}={a,b,q},|$newmin$|<|$newmax$|,W={<{s},{a,b, q}>},T={<{a},0>,<{s},0>}, SSW={(s,q),(a,b)}-{s}※{a, b,q}={(s,q),(a,b)}-{(s,a),(s,b), (s,q)}={(a,b)}. For 1∈Σ, (min□NECSS[1]∨FECSS[1] □min) is false, therefore, ECK[min,1]=min-NECSS[1]={a,b}-{s}={a,b}. For ECKI[min,1]∉T, $newmin$={s,b}, $newmax$={s,a,b,q}-{s,b} ={a,q},|$newmin$|≤|$newmax$|,W={<{s},{a,b,q}>}+{<{a,b}, {a,q}>}={<{s},{a,b,q}>,<{a,b},{a,q}>},T={<{a},0>,<{ s},0>}+{<{a,b},1>,<{q},1>}={<{a},0>,<{s},0>,<{a,b}, 1>,<{q},1>},SSW={(a,b)}-{s,b}※{a,q}={(a,b)}-{(a,s), (a,b),(b,s)}=ø. It can be inferred by SSW==ø that there is no equivalent states. The ninth step is executed to eliminate dead states which will not affect the FA's ability. The minimization result output by the algorithm is: DFA M_{min}=M and this result is same with that of Ref. [4]. The algorithm is fairly efficient for minimizing a finite automaton.

VII. SUMMARY

Based on the concept of CDFA, characteristics of state transition inverse mapping and the usage to reduce repetitive computing are analyzed. The analytical results are used sufficiently to construct a right and generally used DFA minimization algorithm. Further work includes equivalent states recognition based on graphical structure of state transition.

REFERENCES

[1] Alfred V.Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, *Compilers: Principles, Techniques, and Tools, 2nd ed.*. Addison-Wesley, 2007, pp.180-184.

[2] J.E.Hopcroft, "An nlogn algorithm for minimizing the states of a finite automaton," *The Theory of Machines and Computations.* pp.189-196,1971.

[3] Huowang Chen, Chunlin Liu, Qingping Tan, *Programming Language--Compilers Principles, 3rd ed.*. National Defense Industry Press, 2006, pp.56-58.

[4] Shiyang Zhou, Jianhua Zhu, "Study of DFA Minimization," *Computer Engineering and Science.* vol.29, pp.60-62,2007.

[5] Peter Linz, *An Introduction to Formal Languages and Automata, 3rd ed.*. Jones and Bartlett Publishers. 2001, pp.62-67.

[6] Guanghuin Han, Guoli Duan, "Realization of DFA Minimization Algorithm," *Journal of Hubei Industry University.* vol.21, pp.69-71. 2006.