Minimizing Finite Automata with Graph Programs

Detlef Plump\textsuperscript{1}  Robin Suri\textsuperscript{2}  Ambuj Singh\textsuperscript{3}

\textsuperscript{1}The University of York

\textsuperscript{2}Indian Institute of Technology Roorkee

\textsuperscript{3}Indian Institute of Technology Kanpur
Automata Minimization

Problem

Input: A deterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$.
Output: $A' = (Q', \Sigma, \delta', q'_0, F')$ such that $L(A) = L(A')$ and $|Q'|$ is minimal.
Automata Minimization

Problem

Input: A deterministic finite automaton \( A = (Q, \Sigma, \delta, q_0, F) \).
Output: \( A' = (Q', \Sigma, \delta', q'_0, F') \) such that \( L(A) = L(A') \) and \(|Q'|\) is minimal.

Definition

States \( p \) and \( q \) are \textit{equivalent} if for all strings \( w \),

\[ \delta^*(p, w) \in F \text{ if and only if } \delta^*(q, w) \in F. \]
Algorithm of Hopcroft, Motwani and Ullman

Marking phase

Stage 1:
for each $p \in F$ and $q \in Q - F$ do mark the pair $\{p, q\}$

Stage 2:
repeat
  for each non-marked pair $\{p, q\}$ do
    for each $a \in \Sigma$ do
      if $\{\delta(p, a), \delta(q, a)\}$ is marked then mark $\{p, q\}$
  until no new pair is marked
Algorithm of Hopcroft, Motwani and Ullman

Marking phase

Stage 1:

for each $p \in F$ and $q \in Q - F$ do mark the pair $\{p, q\}$

Stage 2:

repeat

for each non-marked pair $\{p, q\}$ do

for each $a \in \Sigma$ do

if $\{\delta(p, a), \delta(q, a)\}$ is marked then mark $\{p, q\}$

until no new pair is marked

Lemma (HMU 07)

Two states $p,q$ are equivalent if and only if the pair $\{p, q\}$ is not marked by the marking phase.
Algorithm of Hopcroft, Motwani and Ullman (cont’d)

Merging phase

Construct $A' = (Q', \Sigma, \delta', q'_0, F')$ as follows:

- $Q' = \{[p] \mid p \in Q\}$, where $[p]$ is the equivalence class of $p$
- $q'_0 = [q_0]$
- $\delta'([p], a) = [\delta(p, a)]$, for each $[p] \in Q'$ and $a \in \Sigma$
- $F' = \{[p] \in Q' \mid [p] \cap F \neq \emptyset\}$
Algorithm of Hopcroft, Motwani and Ullman (cont’d)

Merging phase

Construct $A' = (Q', \Sigma, \delta', q'_0, F')$ as follows:

- $Q' = \{[p] \mid p \in Q\}$, where $[p]$ is the equivalence class of $p$
- $q'_0 = [q_0]$
- $\delta'([p], a) = [\delta(p, a)]$, for each $[p] \in Q'$ and $a \in \Sigma$
- $F' = \{[p] \in Q' \mid [p] \cap F \neq \emptyset\}$

Theorem (HMU 07)

The automaton $A'$ accepts the same language as $A$ and is minimal.
Implementation in GP

Automata are given by their transition diagrams, where

- final/non-final states are labelled $i_1$ resp. $i_0$, with $i \in \mathbb{Z}$,
- only $q_0$ is labelled $1_b$ (with $b \in \{0, 1\}$),
- the symbols in $\Sigma$ are character strings,
- all states are reachable from $q_0$.

![Diagram of automaton with states labeled 1_0, 2_0, 3_0, 4_1 and transitions labeled with 'a' and 'b'.]
Minimization program

```
main = mark; merge; clean_up

mark = distinguish!; propagate!; equate!
merge = init; add_tag!; (choose; add_tag)!;
       disconnect!; redirect!

clean_up = remove_edge!; remove_node!; untag!
```
mark = distinguish!; propagate!; equate!

distinguish(x, y, i, j: int)

\[ x_i \quad \Rightarrow \quad y_j \quad \Rightarrow \quad x_i \quad \Rightarrow \quad y_j \]

where \( i \neq j \) and not edge(1, 2, 1)
Example: execution of distinguish!
mark = distinguish!; propagate!; equate!

\[ \text{propagate}(x, y, u, v, i, j, m, n: \text{int}; s: \text{str}) \]

\[
\begin{array}{ccc}
1 & x_i & 3 \\
& s & u_m \\
& 1 & \\
2 & y_j & 4 \\
& s & v_n
\end{array} \quad \Rightarrow \quad
\begin{array}{ccc}
1 & x_i & 3 \\
& s & u_m \\
& 1 & \\
2 & y_j & 4 \\
& s & v_n
\end{array}
\]

where not edge(1, 2, 1)
all matches
**Quotients of propagate**

\[
\text{propagate}_2(x, u, v, i, m, n: \text{int}; s: \text{str})
\]

\[
\begin{array}{c}
\text{x}_i \\
1 \\
\text{s} \\
\text{v}_n \\
3 \\
\text{s} \\
\text{u}_m \\
1 \\
\end{array} \quad \Rightarrow \quad \begin{array}{c}
\text{x}_i \\
1 \\
\text{s} \\
\text{v}_n \\
3 \\
\text{s} \\
\text{u}_m \\
1 \\
\end{array}
\]

where not edge(1, 4, 1)

\[
\text{propagate}_3(x, u, v, i, m, n: \text{int}; s: \text{str})
\]

\[
\begin{array}{c}
\text{x}_i \\
1 \\
\text{s} \\
\text{v}_n \\
3 \\
\text{s} \\
\text{u}_m \\
1 \\
\end{array} \quad \Rightarrow \quad \begin{array}{c}
\text{x}_i \\
1 \\
\text{s} \\
\text{v}_n \\
3 \\
\text{s} \\
\text{u}_m \\
1 \\
\end{array}
\]

where not edge(1, 3, 1)
Example: execution of propagate!
mark = distinguish!; propagate!; equate!

equate(x, y, i, j: int)

\[ x_i^1 \quad y_j^2 \quad \Rightarrow \quad x_i^1 \quad 0 \quad y_j^2 \]

where not edge(1, 2, 1) and not edge(1, 2, 0)
Example: execution of equate!
merge = init; add_tag!; (choose; add_tag!)!; disconnect!; redirect!

init(i: int)

```
1_i \Rightarrow 1_i_0
```

add_tag(x, y, i, j: int)

```
x_i_0 \rightarrow 0 \quad y_j \Rightarrow x_i_0 \rightarrow 0 \quad y_j_1
```

choose(x, i: int)

```
x_i \Rightarrow x_i_0
```
Example:
execution of \texttt{init; add.tag!; (choose; add.tag)!}
merge = init; add_tag!; (choose; add_tag!)!; disconnect!; redirect!

disconnect(x, u, i, m, p: int; s: str)

all matches
merge = init; add_tag!; (choose; add_tag)!; disconnect!; redirect!

redirect(x, y, u, i, j, m: int; s: str)

all matches
Example: execution of disconnect!; redirect!

![Diagram showing state transitions for disconnect and redirect operations.]

The diagram illustrates the execution sequence of disconnect and redirect operations in a state transition model. The transitions are labeled with symbols representing operations and states, demonstrating the flow and changes in state.
clean_up = remove_edge!; remove_node!; untag!

remove_edge(x,y,i,j,k,m,n: int)

\[ x_{i\,k} \xrightarrow{n} y_{j\,m} \Rightarrow x_{i\,k} \quad y_{j\,m} \]

remove_node(x,i: int)

\[ x_{i\,1} \Rightarrow \emptyset \]

untag(x,i: int)

\[ x_{i\,0} \Rightarrow x_i \]
Example: execution of remove_edge!
Example: execution of `remove_node!; untag!`

```
Example: execution of remove_node!; untag!
```

```
Example: execution of remove_node!; untag!
```
Correctness of the implementation

```
main = mark; merge; clean_up

mark = distinguish!; propagate!; equate!
merge = init; add_tag!; (choose; add_tag)!;
    disconnect!; redirect!
clean_up = remove_edge!; remove_node!; untag!
```

Proposition

*For every input automaton $A$, the minimization program produces an automaton $A'$ that is equivalent to $A$ and minimal.*
Time complexity

Proposition

The minimization program terminates after at most $O(|\Sigma| \cdot |Q|^2)$ rule applications.
Time complexity

Proposition

The minimization program terminates after at most \( O(|\Sigma| \cdot |Q|^2) \) rule applications.

Note: We abstract from the complexity of rule matching.
Complexity of distinguish!

distinguish(x, y, i, j: int)

\[
\begin{array}{c}
\ x_i \\
1
\end{array} \quad \begin{array}{c}
\ y_j \\
2
\end{array} \quad \Rightarrow \quad \begin{array}{c}
\ x_i \\
1
\end{array} \quad \begin{array}{c}
\ y_j \\
2
\end{array}
\]

where \( i \neq j \) and not edge(1, 2, 1)
Complexity of distinguish!

distinguish\((x, y, i, j: \text{int})\)

\[
\begin{array}{ccc}
\text{x}_i & \text{y}_j & \Rightarrow \\
1 & 2 & \\
\end{array}
\]

where \(i \neq j\) and not edge\((1, 2, 1)\)

\[
G \xrightarrow{\text{distinguish}} H \implies \#G > \#H,
\]

where

\[
\#X = |\{\{q, q'\} \mid q \neq q' \land \neg \text{edge}(q, q', 1)\}| < \frac{|Q|^2}{2}
\]
Complexity of propagate!

\[
\text{propagate}(x, y, u, v, i, j, m, n: \text{int}; s: \text{str})
\]

where not edge(1, 2, 1)

all matches
 Complexity of propagate!

propagate\( (x, y, u, v, i, j, m, n: \text{int}; s: \text{str}) \)

\[
\begin{align*}
\quad \quad & x_i \quad s \quad u_m \\
1 & \quad \quad \quad \downarrow \quad 1 \\
\quad \quad & y_j \quad s \quad v_n \\
2 & \quad \quad \quad \downarrow \quad 4 \\
\end{align*}
\]

where not edge\((1, 2, 1)\)
all matches

\[
G \implies H \implies \#G > \#H, \text{ where } \#
\]

\[
\#X = |\{(q, q') | q \neq q' \land \neg \text{edge}(q, q', 1)\}| < \frac{|Q|^2}{2}
\]
Complexity of disconnect!

```
disconnect(x, u, i, m, p: int; s: str)
```

```
2
\( u_m_p \) ⇒ 2
\( u_m_p \)
```

```
1
\( x_{i1} \) ⇒ 1
\( x_{i1} \)
```

all matches
Complexity of disconnect!

\[ \text{disconnect}(x, u, i, m, p: \text{int}; s: \text{str}) \]

all matches

\[ G \xrightarrow{\text{disconnect}} H \] implies \( \#G > \#H \), where

\[ \#X = \left| \{ e \in E_X \mid \text{lab}(e) \in \Sigma \} \right| \leq |\Sigma| \cdot |Q|^2 \]
Complexity of redirect!

```
redirect(x, y, u, i, j, m: int; s: str)
```

all matches
Complexity of redirect!

```
redirect(x, y, u, i, j, m: int; s: str)
```

all matches

\[
G \xrightarrow{\text{redirect}} H \text{ implies } \#G > \#H,
\]

where

\[
\#X = |\{ e \in E_X \mid \text{lab}(e) \in \Sigma \land \text{lab}(	ext{target}(e)) = x.i_1 \}| \leq |\Sigma| \cdot |Q|^2
\]
Conclusion

- Minimizing automata with rule-based, visual programming
- Direct manipulation of transition diagrams (augmented with auxiliary labels and edges); no need to implement transition functions, state tables, etc.
- Rule schemata and control constructs allow formal reasoning at a high level of abstraction
- Case study reveals convenience/necessity of certain GP constructs, e.g. ‘all matches’ attribute and ternary edge predicate
Possible extensions

- Implementing state merging by non-injective rules — requires GP extension
- Implementing more efficient minimization algorithms such as the $n^2$ algorithm of Hopcroft and Ullman or the $n \log n$ algorithm of Hopcroft
- Minimizing nondeterministic finite automata