Reducing a DFA to a Minimal DFA

**Input:**
DFA\_IN
Assume DFA\_IN never “gets stuck”
(add a dead state if necessary)

**Output:**
DFA\_MIN
An equivalent DFA with the minimum number of states.
Reducing a DFA to a Minimal DFA

**Input:**
DFA\(_{IN}\)
Assume DFA\(_{IN}\) never “gets stuck”
(add a dead state if necessary)

**Output:**
DFA\(_{MIN}\)
An equivalent DFA with the minimum number of states.

**Approach:**
Merge two states if the effectively do the same thing.
“Do the same thing?”
At EOF, is DFA\(_{IN}\) in an accepting state or not?

Sufficiently Different States
Merge states, if at all possible.

Are two states “sufficiently different”
... that they cannot be merged?
**Sufficiently Different States**

Merge states, if at all possible.

Are two states “*sufficiently different*”
... that they cannot be merged?

State $s$ is “distinguished” from state $t$ by some string $w$ iff:

- starting at $s$, given characters $w$, the DFA ends up accepting,
- ... but starting at $t$, the DFA does not accept.

---

**Example:**

```
S -> a -> c -> s

S -> b -> c -> t

t
```

“$ab$” does not distinguish $s$ and $t$.
But “$c$” distinguishes $s$ and $t$.
Therefore, $s$ and $t$ cannot be merged.
Partitioning a Set

A partitioning of a set...
...breaks the set into non-overlapping subsets.
(The partition breaks the set into “groups”)

Example:
S = \{A, B, C, D, E, F, G\}
\Pi = \{(A) \ (B \ C \ D \ E \ F \ G)\}
\Pi_2 = \{(A) \ (B \ C) \ (D \ E \ F \ G)\}

Note:
\{ (...) (...) (...) \} means \{ {...}, {...}, {...} \}
Hopcroft’s Algorithm

Consider the set of states.

Partition it...
- Final States
- All Other States

Repeatedly “refine” the partitioning.
Two states will be placed in different groups
... If they can be “distinguished”

Repeat until no group contains states that can be distinguished.

Each group in the partitioning becomes one state in a newly constructed DFA
\[ \text{DFA}_{\text{MIN}} = \text{The minimal DFA} \]

How to Refine a Partitioning?

\[ \Pi_1 = \{ (A B D), (C E) \} \]

Consider one group... (A B D)
Look at output edges on some symbol (e.g., “x”)
How to Refine a Partitioning?

\[ \Pi_1 = \{ (A, B, D), (C, E) \} \]

Consider one group... \((A, B, D)\)
Look at output edges on some symbol (e.g., “\(x\)”) on “\(x\)”, all states in \(P_1\) go to states belonging to the same group.

Now consider another symbol (e.g., “\(y\)”)
How to Refine a Partitioning?

\[ \Pi_1 = \{ (A \ B \ D) \ (C \ E) \} \]

Consider one group... (A B D)
Look at output edges on some symbol (e.g., “x”)

On “x”, all states in \( P_1 \) go to states belonging to the same group.

Now consider another symbol (e.g., “y”)
D is distinguished from A and B!

So refine the partition!
\[ \Pi_{i+1} = \{ (A \ B) \ (D) \ (C \ E) \} \]
Example

Initial Partitioning: $\Pi_1 = (A \ B \ C \ D) \ (E)$

Consider (A B C D)

Consider (E)
**Example**

Initial Partitioning: $\Pi_1 = (A \ B \ C \ D) \ (E)$

- Consider $(A \ B \ C \ D)$
  - Consider “$a$”
  - Consider “$b$”
- Consider $(E)$

---

**Example**

Initial Partitioning: $\Pi_1 = (A \ B \ C \ D) \ (E)$

- Consider $(A \ B \ C \ D)$
  - Consider “$a$”
    - Break apart?
  - Consider “$b$”
- Consider $(E)$
Example

Initial Partitioning: $\Pi_1 = (A \ B \ C \ D) \ (E)$
Consider $(A \ B \ C \ D)$
  Consider “a”
    Break apart? No
  Consider “b”
    Break apart?
Consider $(E)$
Example

Initial Partitioning: \( \Pi_1 = (A \ B \ C \ D) \ (E) \)
Consider \((A \ B \ C \ D)\)
   Consider “a”
   Break apart? No
   Consider “b”
   Break apart? \((A \ B \ C) \ (D)\)
Consider \((E)\)
   Not possible to break apart.

Example

Initial Partitioning: \( \Pi_1 = (A \ B \ C \ D) \ (E) \)
Consider \((A \ B \ C \ D)\)
   Consider “a”
   Break apart? No
   Consider “b”
   Break apart? \((A \ B \ C) \ (D)\)
Consider \((E)\)
   Not possible to break apart.
New Partitioning: \( \Pi_2 = (A \ B \ C) \ (D) \ (E)\)
Example
Initial Partitioning: $\Pi_1 = (A \ B \ C \ D) \ (E)$
Consider $(A \ B \ C \ D)$
  Consider “a”
    Break apart? No
  Consider “b”
    Break apart? $(A \ B \ C) \ (D)$
Consider (E)
  Not possible to break apart.
New Partitioning: $\Pi_2 = (A \ B \ C) \ (D) \ (E)$
  Consider “a”
  Break apart?
  Consider “b”

Example
Initial Partitioning: $\Pi_1 = (A \ B \ C \ D) \ (E)$
Consider $(A \ B \ C \ D)$
  Consider “a”
    Break apart? No
  Consider “b”
    Break apart? $(A \ B \ C) \ (D)$
Consider (E)
  Not possible to break apart.
New Partitioning: $\Pi_2 = (A \ B \ C) \ (D) \ (E)$
  Consider “a”
  Break apart? No
  Consider “b”
  Break apart?
**Example**

Initial Partitioning: $\Pi_1 = (A \ B \ C \ D) \ (E)$
Consider $(A \ B \ C \ D)$
  Consider “$a$”
    Break apart? No
  Consider “$b$”
    Break apart? $(A \ B \ C) \ (D)$
Consider $(E)$
  Not possible to break apart.
New Partitioning: $\Pi_2 = (A \ B \ C) \ (D) \ (E)$
  Consider “$a$”
    Break apart? No
  Consider “$b$”
    Break apart? $(A \ C) \ (B)$

---

New Partitioning: $\Pi_3 = (A \ C) \ (B) \ (D) \ (E)$
  Consider “$a$”
    Break apart?
  Consider “$b$”
    Break apart?
Example

Initial Partitioning: $\Pi_1 = (A \ B \ C \ D) \ (E)$
Consider (A B C D)
  Consider “a”
    Break apart? No
  Consider “b”
    Break apart? (A B C) (D)
Consider (E)
    Not possible to break apart.
New Partitioning: $\Pi_2 = (A \ B \ C) \ (D) \ (E)$
Consider “a”
    Break apart? No
Consider “b”
    Break apart? (A C) (B)
New Partitioning: $\Pi_3 = (A \ C) \ (B) \ (D) \ (E)$
Consider “a”
    Break apart? No
Consider “b”
    Break apart?
Example

Initial Partitioning: $\Pi_1 = (A \ B \ C \ D) \ (E)$
Consider (A B C D)
  
  Consider “a”
  
  Break apart? No
Consider “b”
  
  Break apart? (A B C) (D)
Consider (E)
  
  Not possible to break apart.

New Partitioning: $\Pi_2 = (A \ B \ C) \ (D) \ (E)$
Consider “a”
  
  Break apart? No
Consider “b”
  
  Break apart? (A C) (B)

New Partitioning: $\Pi_3 = (A \ C) \ (B) \ (D) \ (E)$
Consider “a”
  
  Break apart? No
Consider “b”
  
  Break apart? No

Hopcroft’s Algorithm

Add dead state and transitions to it if necessary.
(Now, every state has an outgoing edge on every symbol.)

$\Pi = \text{initial partitioning}$

loop
  $
  \Pi_{\text{NEW}} = \text{Refine}(\Pi)
  $
  
  if ($\Pi_{\text{NEW}} = \Pi$) then break

  $\Pi = \Pi_{\text{NEW}}$
endLoop
Hopcroft’s Algorithm

Add dead state and transitions to it if necessary. 
(Now, every state has an outgoing edge on every symbol.)

\[ \Pi = \text{initial partitioning} \]

\[ \text{loop} \]
\[ \Pi_{\text{NEW}} = \text{Refine}(\Pi) \]
\[ \text{if} \ (\Pi_{\text{NEW}} = \Pi) \ \text{then break} \]
\[ \Pi = \Pi_{\text{NEW}} \]
\[ \text{endLoop} \]

Construct \( \text{DFA}_{\text{MIN}} \)
- Each group in \( \Pi \) becomes a state
Hopcroft’s Algorithm

Add dead state and transitions to it if necessary.
(Now, every state has an outgoing edge on every symbol.)

\[ \Pi = \text{initial partitioning} \]

\[ \text{loop} \]
\[ \Pi_{\text{NEW}} = \text{Refine}(\Pi) \]
\[ \text{if } (\Pi_{\text{NEW}} = \Pi) \text{ then break} \]
\[ \Pi = \Pi_{\text{NEW}} \]
\[ \text{endLoop} \]

Construct \( \text{DFA}_{\text{MIN}} \)
- Each group in \( \Pi \) becomes a state
- Choose one state in each group
  (throw all other states away)
- Preserve the edges out of the chosen state

© Harry H. Porter, 2005

Lexical Analysis - Part 4
Hopcroft’s Algorithm

Add dead state and transitions to it if necessary.
(Now, every state has an outgoing edge on every symbol.)

\[ \Pi = \text{initial partitioning} \]
\[ \text{loop} \]
\[ \Pi_{\text{NEW}} = \text{Refine}(\Pi) \]
\[ \text{if } (\Pi_{\text{NEW}} = \Pi) \text{ then break} \]
\[ \Pi = \Pi_{\text{NEW}} \]
\[ \text{endLoop} \]

Construct DFA_{MIN}
- Each group in \( \Pi \) becomes a state
- Choose one state in each group
  (throw all other states away)
- Preserve the edges out of the chosen state
- Deal with start state and final states
Hopcroft’s Algorithm

Add dead state and transitions to it if necessary. (Now, every state has an outgoing edge on every symbol.)

\[ \Pi = \text{initial partitioning} \]
\[ \text{loop} \]
\[ \Pi_{\text{NEW}} = \text{Refine}(\Pi) \]
\[ \text{if } (\Pi_{\text{NEW}} = \Pi) \text{ then break} \]
\[ \Pi = \Pi_{\text{NEW}} \]
\[ \text{endLoop} \]

Construct \( \text{DFA}_{1}\)
- Each group in \( \Pi \) becomes a state
- Choose one state in each group (throw all other states away)
- Preserve the edges out of the chosen state
- Deal with start state and final states
- If desired...
  - Remove dead state
  - Remove any state unreachable from the start state

\[ \Pi_{\text{NEW}} = \text{Refine}(\Pi) \]

\[ \Pi_{\text{NEW}} = {} \]
\[ \text{for each group } G \text{ in } \Pi \text{ do} \]
\[ \text{Example: } \Pi = (A B C E) (D F) \]
\[ \text{Break } G \text{ into sub-groups} \]
\[ (A B C E) \rightarrow (A C) (B E) \]
\[ \text{as follows:} \]
- Put \( S \) and \( T \) into different subgroups if...
- For any symbol \( a \in \Sigma \), \( S \) and \( T \) go to states in two different groups in \( \Pi \)

\[ \begin{align*}
A & \xrightarrow{x} D \\
B & \xrightarrow{x} C \\
\end{align*} \]

\[ \text{Add the sub-groups to } \Pi_{\text{NEW}} \]
\[ \text{return } \Pi_{\text{NEW}} \]
• Regular Expressions to Describe Tokens
Summarizing...

- Regular Expressions to Describe Tokens
- Algorithm: Regular Expression → NFA
- Algorithm for Simulating NFA
Lexical Analysis - Part 4

Summarizing...

• Regular Expressions to Describe Tokens
• Algorithm: Regular Expression → NFA
• Algorithm for Simulating NFA
• Algorithm: NFA → DFA

Lexical Analysis - Part 4

Summarizing...

• Regular Expressions to Describe Tokens
• Algorithm: Regular Expression → NFA
• Algorithm for Simulating NFA
• Algorithm: NFA → DFA
• Algorithm: DFA → Minimal DFA
Lexical Analysis - Part 4

**Summarizing...**

- Regular Expressions to Describe Tokens
- Algorithm: Regular Expression → NFA
- Algorithm for Simulating NFA
- Algorithm: NFA → DFA
- Algorithm: DFA → Minimal DFA
- Algorithm for Simulating DFA

**Fast:**
- Get Next Char
- Evaluate Move Function
e.g., Array Lookup
- Change State Variable
- Test for Accepting State
- Test for EOF
- Repeat
**Summarizing...**

- Regular Expressions to Describe Tokens
- Algorithm: Regular Expression $\rightarrow$ NFA
- Algorithm for Simulating NFA
- Algorithm: NFA $\rightarrow$ DFA
- Algorithm: DFA $\rightarrow$ Minimal DFA
- Algorithm for Simulating DFA
  - **Fast:**
    - Get Next Char
    - Evaluate Move Function
      - e.g., Array Lookup
    - Change State Variable
    - Test for Accepting State
    - Test for EOF
    - Repeat

- Scanner Generators

*Create an efficient Lexer from regular expressions!*

---

**Scanner Generator: LEX**

*Input:*

$r_1$ { action$_1$ }
$r_2$ { action$_2$ }
...
$r_N$ { action$_N$ }

*Requirements:*

- Choose the largest lexeme that matches.
- If more than one $r_i$ matches, choose the first one.

---

**DFA Simulator (C-code)**

- Transition Tables (initialized arrays)

**Input Buffers**

- `lex-begin-ptr`
- `forward-ptr`

*“Canned code” added by lex tool*

*Computed by lex tool*
Input:
- a  { Action-1 }
- abb { Action-2 }
- a*b+ { Action-3 }

Create NFA:
- a | abb | a*b+
**Input:**
- \(a\) { Action-1 }
- \(abb\) { Action-2 }
- \(a*b+\) { Action-3 }

Create NFA:
- \(a \mid abb \mid a*b+\)

**Example Input:** “aabc...”
- Start: \(\{0, 1, 3, 7\}\)
- Input: “\(a\)”
  - \(\{2, 4, 7\}\)
- Input: “\(a\)”
  - \(\{7\}\)
- Input: “\(b\)”
  - \(\{8\}\)
- Input: “\(c\)”
  - \(\{}\)

Match!  
Pattern: 1  
Length: 1

Match!  
Pattern: 3  
Length: 3

Done!  
Identify the last match.  
Execute the corresponding action & adjust pointers

---

**Approach**

- Find the NFA for  
  \(r_1 \mid r_2 \mid \ldots \mid r_N\)
- Convert to a DFA.
- Each state of the DFA corresponds to a set of NFA states.
- A state is final if any NFA state in it was a final state.
- If several, choose the lowest numbered pattern to be the one accepted.
- During simulation, keep following edges until you get stuck.
- As the scanning proceeds...
  Every time you enter a final state...
  Remember:
  The current value of buffer pointers  
  Which pattern was recognized
- Upon termination...
  Use that information to...
  Adjust the buffer pointers  
  Execute the desired action
Example

Input:
- a { Action-1 }
- abb { Action-2 }
- a*b+ { Action-3 }

Create NFA:
- a | abb | a*b+
Example

Input:
- a { Action-1 }
- abb { Action-2 }
- a*b+ { Action-3 }

Create NFA:
- a | abb | a*b+

Construct Minimal DFA

Attach Actions

Accept only first pattern
**Example**

**Input:**

\[
\begin{align*}
\text{a} & \quad \{ \text{Action-1} \} \\
\text{abb} & \quad \{ \text{Action-2} \} \\
\text{a*b+} & \quad \{ \text{Action-3} \}
\end{align*}
\]

**Create NFA:**

\[
\begin{align*}
\text{a} & \mid \text{abb} \mid \text{a*b+}
\end{align*}
\]

**Construct Minimal DFA**

**Attach Actions**

**Example Strings:**

\[
\begin{align*}
\text{a} \\
\text{ab} \\
\text{abbbbb} \\
\text{abb}
\end{align*}
\]

**The “Lex” Tool**

Oldest, most well-known
For Unix/C Environment

**In UNIX:**

\[
\begin{align*}
\% \text{lex lex.l} \\
\% \text{cc lex.yy.c}
\end{align*}
\]

File: “*lex.l***

**Lex Tool**

File: “*lex.yy.c***

Contains several regular expressions

A program in “C”...
Ready to compile and link with Parser (e.g., YACC output)
Lexical Analysis - Part 4

The “Lex” Tool

Oldest, most well-known
For Unix/C Environment

In UNIX:

```
% lex lex.l
% cc lex.yy.c
```

Input File Format:

```
{%
    ...
} %
    ...
    %
    ...
%
    ...
%
    ...
%
```

File: “lex.l”

File: “lex.yy.c”

Contains several regular expressions

A program in “C”...
Ready to compile and link with Parser (e.g., YACC output)

Regular Expressions in Lex

abc Concatenation; Most characters stand for themselves

Meta Characters:

| | Usual meanings
* | Example: (a|b)*c*
() | One or more, e.g., ab+c
? | Optional, e.g., ab?c
[x-y] | Character classes, e.g., [a-z][a-zA-Z0-9]*
[^x-y] | Anything but [x-y]
\ | The usual escape sequences, e.g., \n.
\ | Any character except ‘\n’
\ | Beginning of line
$ | End of line
" ... " | To use the meta characters literally,
            Example: PCAT comments: " (*".*") "
{ ... } | Defined names, e.g., {letter}
/ | Look-ahead
    Example: ab/cd
    (Matches ab, but only when followed by cd)
Look-Ahead Operator, /

`abb/cd`

“Matches `abb`, but only if followed by `cd`.”

Add a special `ε` edge for `/`
Look-Ahead Operator, /

`abb/cd`

“Matches `abb`, but only if followed by `cd`.”

Add a special `ε` edge for `/`

Mark the following state to make a note of...

- The pattern in question
- The current value of the buffer pointers

...whenever this state is encountered during scanning.

When a pattern is finally matched, check these notes.

- If we passed through a “/” state for the pattern accepted,
  Use the stored buffer positions,
  instead of the final positions to describe the lexeme matched.
Lex: Input File Format

{%
...Any “C” Code...
%
...Regular Definitions...
%
...Regular Expressions with Actions...
%
...Any “C” Code...

#define ID 13
#define NUM 14
#define PLUS 15
#define MINUS 16
...
#define WHILE 37
#define IF 38
...
%
...Regular Definitions...
%
...Regular Expressions with Actions...
%
...Any “C” Code...

int lookup (char * p) {...}
int enter (char * p, int i) {...}
...
Lex: Input File Format

{%

...Any “C” Code...
%

...Regular Definitions...

\%

}\

...Regular Expressions with Actions...

\%

...Any “C” Code...

---

Lex: Input File Format

{%

...Any “C” Code...
%

...Regular Definitions...

}\%

...Regular Expressions with Actions...

...Any “C” Code...

---

Lex: Input File Format

{%

...Any “C” Code...
%

...Regular Definitions...

\%

}\

...Regular Expressions with Actions...

...Any “C” Code...

---

Lex: Input File Format

{%

...Any “C” Code...
%

...Regular Definitions...

}\%

...Regular Expressions with Actions...

...Any “C” Code...

---