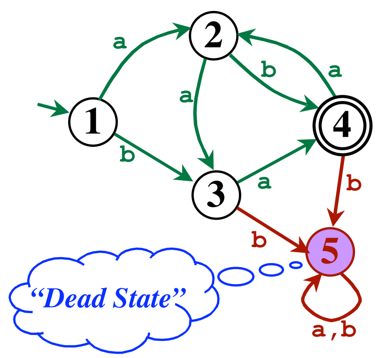
Reducing a DFA to a Minimal DFA

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

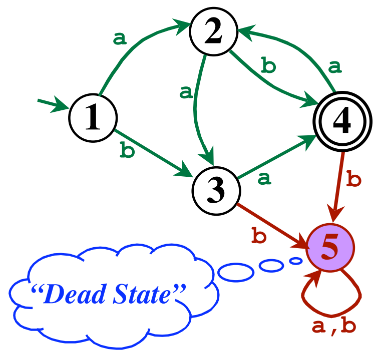
Input: DFA_{IN}
Assume DFA_{IN} never "gets stuck"
(add a dead state if necessary)



Output: DFA_{MIN}
An equivalent DFA with the minimum number of states.

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.

Lexical Analysis - Part 4

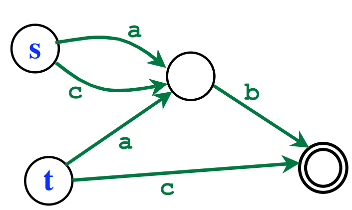
Sufficiently Different States

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:



"`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:

$S = \{A, B, C, D, E, F, G\}$

$\Pi = \{(A\ B)\ (C\ D\ E\ F)\ (G)\ \}$

$\Pi_2 = \{(A)\ (B\ C)\ (D\ E\ F\ G)\ \}$

Partitioning a Set

A partitioning of a set...

...breaks the set into non-overlapping subsets.

(The partition breaks the set into "groups")

Example:

$S = \{A, B, C, D, E, F, G\}$

$\Pi = \{(A\ B)\ (C\ D\ E\ F)\ (G)\ \}$

$\Pi_2 = \{(A)\ (B\ C)\ (D\ E\ F\ G)\ \}$

We can "refine" a partition...

$\Pi_i = \{\ (A\ B\ C)\quad (D\ E)\quad (F\ G)\ \}$

$\Pi_{i+1} = \{\ (A\ C)\quad (B)\quad (D)\quad (E)\quad (F\ G)\ \}$

Note:

{ (...) (...) (...) } means {{...}, {...}, {...} }

Hopcroft's Algorithm

Consider the set of states.

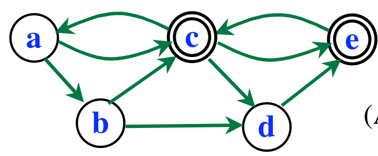
Partition it...

- Final States
- All Other States

Repeatedly "refine" the partioning.

Two states will be placed in different groups

... If they can be "distinguished"



(A B D) (C E)

Repeat until no group contains states that can be distinguished.

Each group in the partitioning becomes one state in a newly constructed DFA

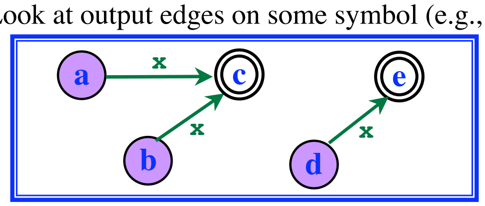
DFA_{MIN} = The minimal DFA

How to Refine a Partitioning?

$$\Pi_i = \{\ \underbrace{(A\ \ B\ \ D)}_{P_1}\ \ \underbrace{(C\ \ E)}_{P_2}\ \}$$

Consider one group... (A B D)

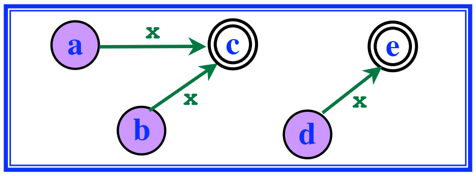
Look at output edges on some symbol (e.g., "**x**")



How to Refine a Partitioning?

$$\Pi_i = \{\ \underbrace{(A\ \ B\ \ D)}_{P_1}\ \ \underbrace{(C\ \ E)}_{P_2}\ \}$$

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

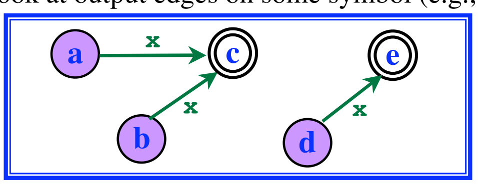


On "**x**", all states in P_1 go to states belonging to the same group.

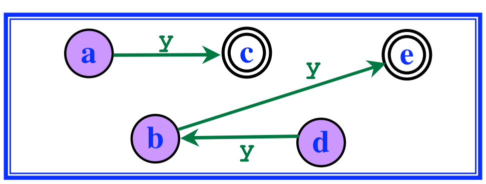
How to Refine a Partitioning?

$$\Pi_i = \{\ \underbrace{(A\ \ B\ \ D)}_{P_1}\ \ \underbrace{(C\ \ E)}_{P_2}\ \}$$

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.

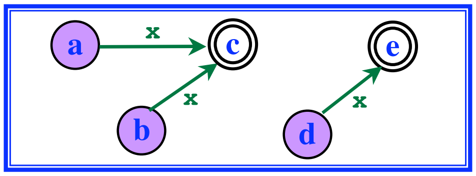


Now consider another symbol (e.g., "**y**")

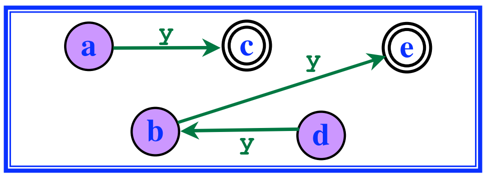
How to Refine a Partitioning?

$\Pi_i = \{ \underbrace{(A\ B\ D)}_{P_1}\ \underbrace{(C\ E)}_{P_2} \}$

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.

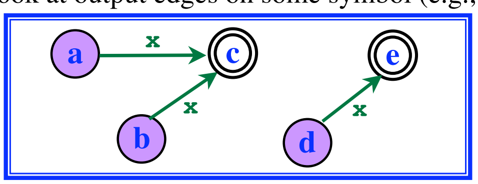


Now consider another symbol (e.g., "**y**")
D is distinguished from A and B!

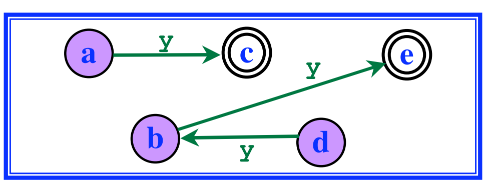
How to Refine a Partitioning?

$\Pi_i = \{ \underbrace{(A\ B\ D)}_{P_1}\ \underbrace{(C\ E)}_{P_2} \}$

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



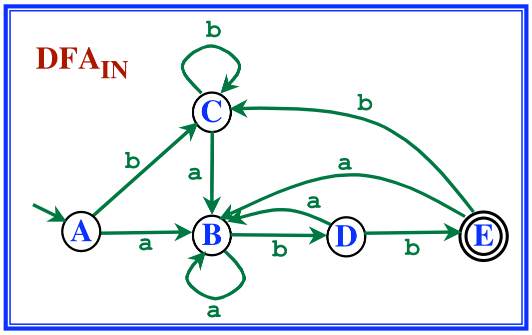
On "**x**", all states in P_1 go to states belonging to the same group.



Now consider another symbol (e.g., "**y**")
D is distinguished from A and B!
So refine the partition!
$\Pi_{i+1} = \{ \underbrace{(A\ B)}_{P_3}\ \underbrace{(D)}_{P_4}\ \underbrace{(C\ E)}_{P_2} \}$

Example

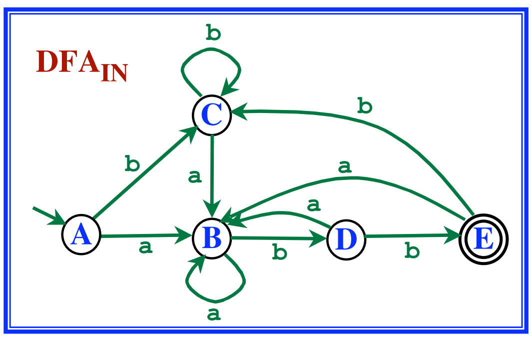
Initial Partitioning: Π_1 = (A B C D) (E)



Lexical Analysis - Part 4

Example

Initial Partitioning: Π_1 = (A B C D) (E)
Consider (A B C D)



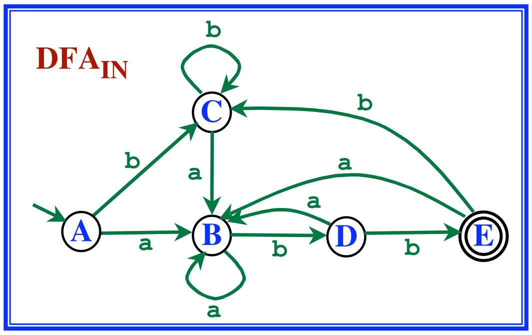
Consider (E)

Example

Initial Partitioning: Π_1 = (A B C D) (E)
Consider (A B C D)
Consider "**a**"

Consider "**b**"

Consider (E)

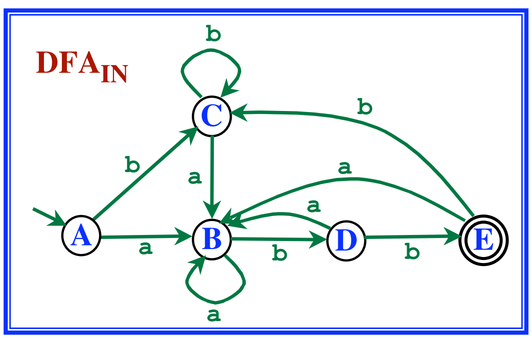


Lexical Analysis - Part 4

Example

Initial Partitioning: Π_1 = (A B C D) (E)
Consider (A B C D)
Consider "**a**"
Break apart?
Consider "**b**"

Consider (E)



Example

Initial Partitioning: Π_1 = (A B C D) (E)

Consider (A B C D)

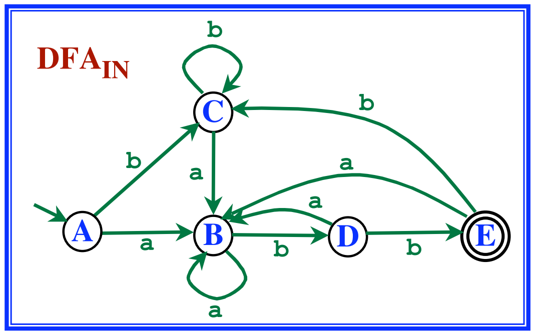
Consider “**a**”

Break apart? No

Consider “**b**”

Break apart?

Consider (E)



Lexical Analysis - Part 4

Example

Initial Partitioning: Π_1 = (A B C D) (E)

Consider (A B C D)

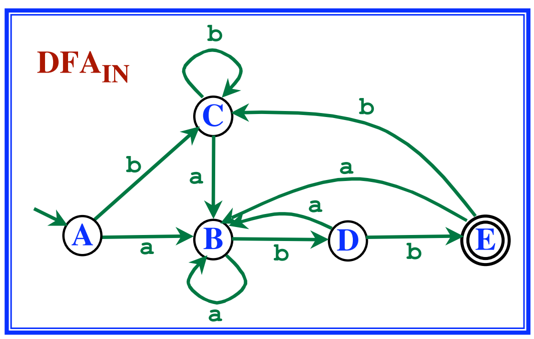
Consider “**a**”

Break apart? No

Consider “**b**”

Break apart? (A B C) (D)

Consider (E)



Example

Initial Partitioning: Π_1 = (A B C D) (E)

Consider (A B C D)

Consider "**a**"

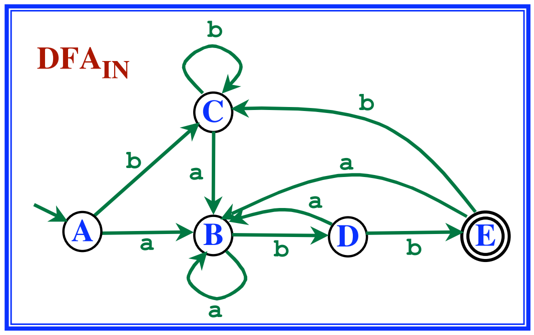
Break apart? No

Consider "**b**"

Break apart? (A B C) (D)

Consider (E)

Not possible to break apart.



Example

Initial Partitioning: Π_1 = (A B C D) (E)

Consider (A B C D)

Consider "**a**"

Break apart? No

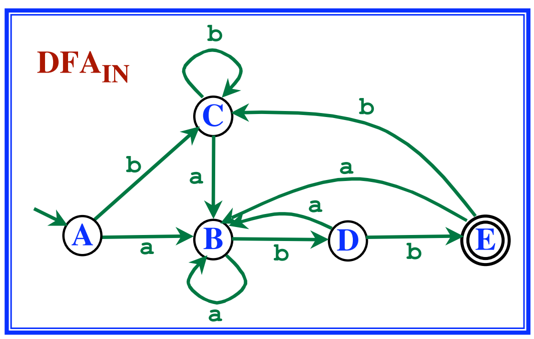
Consider "**b**"

Break apart? (A B C) (D)

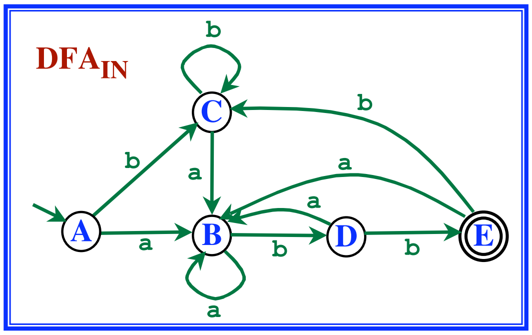
Consider (E)

Not possible to break apart.

New Partitioning: Π_2 = (A B C) (D) (E)

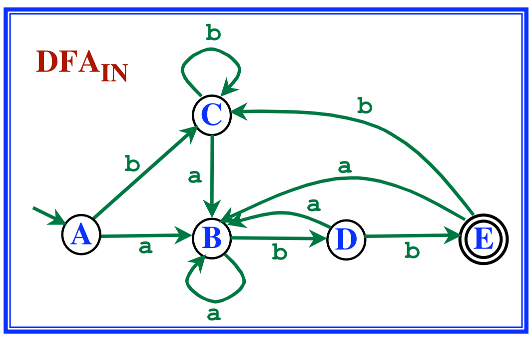


Example



Initial Partitioning: Π_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: Π_2 = (A B C) (D) (E)
Consider "**a**"
Break apart?
Consider "**b**"

Example



Initial Partitioning: Π_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: Π_2 = (A B C) (D) (E)
Consider "**a**"
Break apart? No
Consider "**b**"
Break apart?

Example

Initial Partitioning: Π_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.

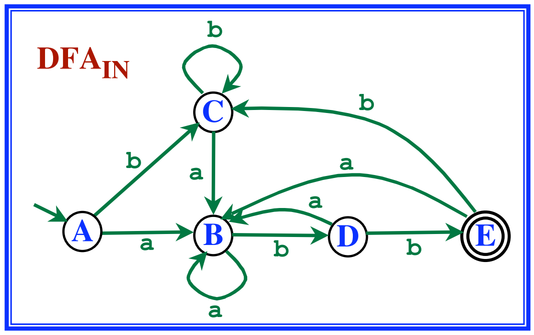
New Partitioning: Π_2 = (A B C) (D) (E)

Consider "**a**"

Break apart? No

Consider "**b**"

Break apart? (A C) (B)



Example

Initial Partitioning: Π_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: Π_2 = (A B C) (D) (E)

Consider "**a**"

Break apart? No

Consider "**b**"

Break apart? (A C) (B)

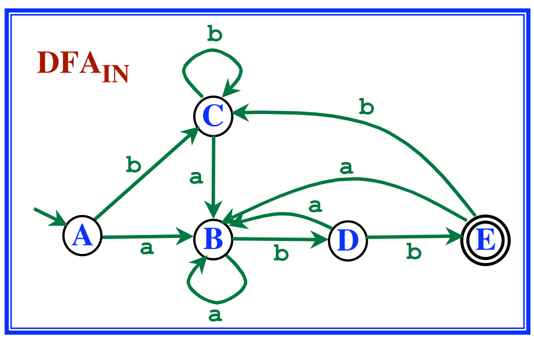
New Partitioning: Π_3 = (A C) (B) (D) (E)

Consider "**a**"

Break apart?

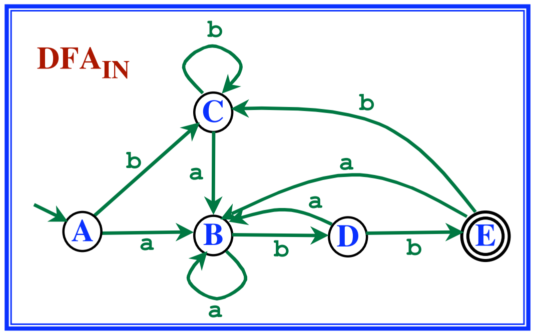
Consider "**b**"

Break apart?



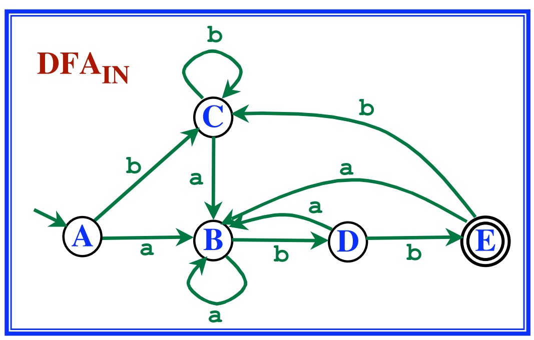
Example

Initial Partitioning: Π_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: Π_2 = (A B C) (D) (E)
Consider "**a**"
Break apart? No
Consider "**b**"
Break apart? (A C) (B)
New Partitioning: Π_3 = (A C) (B) (D) (E)
Consider "**a**"
Break apart? No
Consider "**b**"
Break apart?



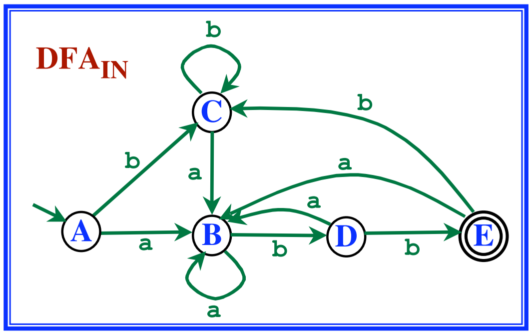
Example

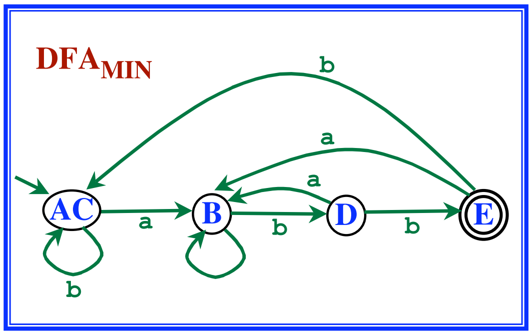
Initial Partitioning: Π_1 = (A B C D) (E)
Consider (A B C D)
Consider "**a**"
Break apart? No
Consider "**b**"
Break apart? (A B C) (D)
Consider (E)
Not possible to break apart.
New Partitioning: Π_2 = (A B C) (D) (E)
Consider "**a**"
Break apart? No
Consider "**b**"
Break apart? (A C) (B)
New Partitioning: Π_3 = (A C) (B) (D) (E)
Consider "**a**"
Break apart? No
Consider "**b**"
Break apart? No



Example

Initial Partitioning: Π_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: Π_2 = (A B C) (D) (E)
Consider "**a**"
Break apart? No
Consider "**b**"
Break apart? (A C) (B)
New Partitioning: Π_3 = (A C) (B) (D) (E)
Consider "**a**"
Break apart? No
Consider "**b**"
Break apart? No





Hopcroft's Algorithm

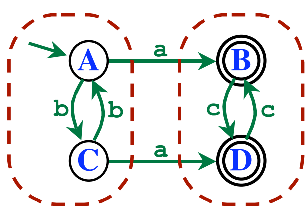
```
Add dead state and transitions to it if necessary.
   (Now, every state has an outgoing edge on every symbol.)

Π = initial partitioning
loop
  Π_NEW = Refine(Π)
  if (Π_NEW = Π) then break
  Π = Π_NEW
endLoop
```

Hopcroft's Algorithm

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

Π = initial partitioning
loop
  Π_NEW = Refine(Π)
  if (Π_NEW = Π) then break
  Π = Π_NEW
endLoop

Construct DFA_MIN
  • Each group in Π becomes a state
```

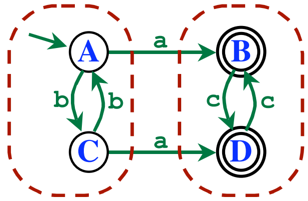

Hopcroft's Algorithm

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

Π = initial partitioning
loop
  Π_NEW = Refine(Π)
  if (Π_NEW = Π) then break
  Π = Π_NEW
endLoop

Construct DFA_MIN
  • Each group in Π becomes a state
```



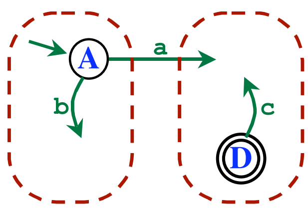
Hopcroft's Algorithm

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

Π = initial partitioning
loop
  Π_NEW = Refine(Π)
  if (Π_NEW = Π) then break
  Π = Π_NEW
endLoop

Construct DFA_MIN
  • Each group in Π becomes a state
  • Choose one state in each group
    (throw all other states away)
  • Preserve the edges out
            of the chosen state
```



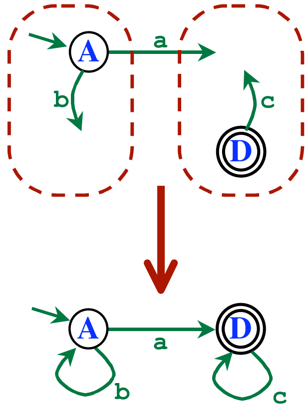
Hopcroft's Algorithm

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

Π = initial partitioning
loop
  Π_NEW = Refine(Π)
  if (Π_NEW = Π) then break
  Π = Π_NEW
endLoop

Construct DFA_MIN
  • Each group in Π becomes a state
  • Choose one state in each group
    (throw all other states away)
  • Preserve the edges out
            of the chosen state
```



Hopcroft's Algorithm

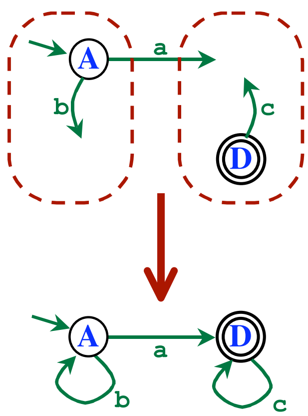
```
Add dead state and transitions to it if necessary.
   (Now, every state has an outgoing edge on every symbol.)

Π = initial partitioning
loop
  Π_NEW = Refine(Π)
  if (Π_NEW = Π) then break
  Π = Π_NEW
endLoop

Construct DFA_MIN
  • Each group in Π becomes a state
  • Choose one state in each group
    (throw all other states away)
  • Preserve the edges out
             of the chosen state
```



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.)

Π = initial partitioning
loop
  Π_NEW = Refine(Π)
  if (Π_NEW = Π) then break
  Π = Π_NEW
endLoop

Construct DFA_MIN
  • Each group in Π 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
```

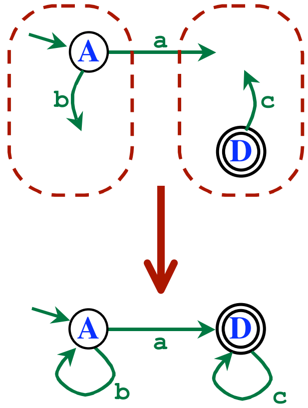


Hopcroft's Algorithm

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

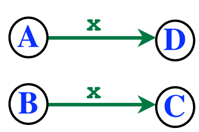
Π = initial partitioning
loop
  Π_{NEW} = Refine(Π)
  if (Π_{NEW} = Π) **then** **break**
  Π = Π_{NEW}
endLoop

Construct DFA_{MIN}
- Each group in Π 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



Π_{NEW} = Refine(Π)

Π_{NEW} = {}
for each group G in Π **do**
  Example: Π = (A B C E) (D F)
  Break G into sub-groups
  (A B C E) → (A C) (B E)
  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 Π



Must split A and B into different groups

  Add the sub-groups to Π_{NEW}
endFor
return Π_{NEW}

Π_{NEW} = { }
Π_{NEW} = { (A C) (B E) }
Π_{NEW} = { (A C) (B E) (D F) }

Summarizing...

Summarizing...

- Regular Expressions to Describe Tokens

Summarizing...

- Regular Expressions to Describe Tokens
- Algorithm: Regular Expression → NFA

Lexical Analysis - Part 4

Summarizing...

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

Summarizing...

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

Summarizing...

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

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

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 → 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
- Scanner Generators

  Create an efficient Lexer from regular expressions!

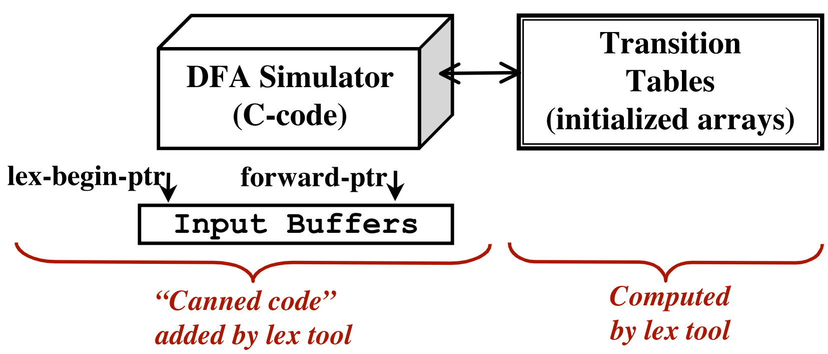
Scanner Generator: LEX

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.



Input:

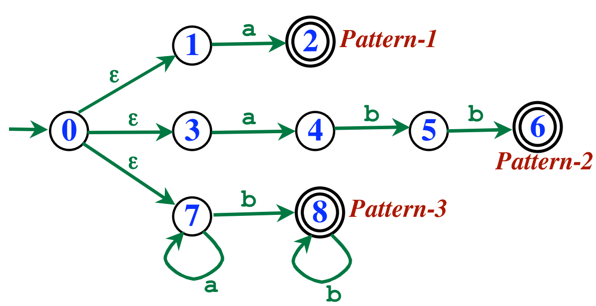
`a`	{ Action-1 }
`abb`	{ Action-2 }
`a*b+`	{ Action-3 }

Input:

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

Create NFA:

`a | abb | a*b+`

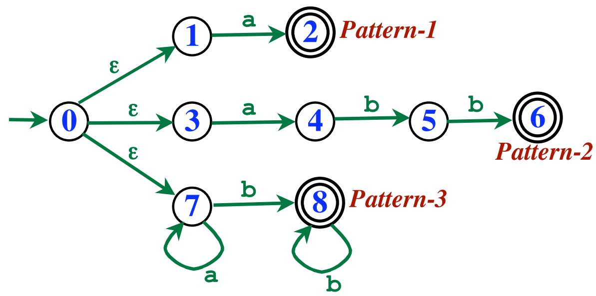


Input:

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

Create NFA:

```
a | abb | a*b+
```



Example Input: `"aabc..."`

Start: { 0, 1, 3, 7 }
Input: `"a"`
{ **2**, 4, 7 }

Match!
Pattern: 1
Length: 1

Input: `"a"`
{ 7 }
Input: `"b"`
{ **8** }

Match!
Pattern: 3
Length: 3

Input: `"c"`
{ }

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 }

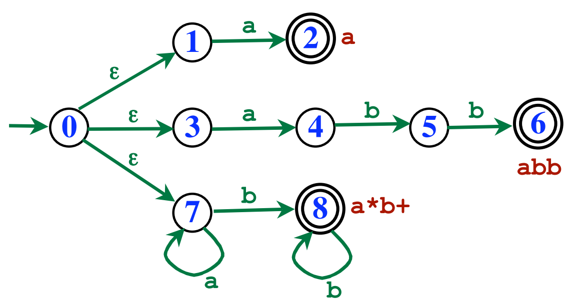
Example

Input:

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

Create NFA:

`a | abb | a*b+`



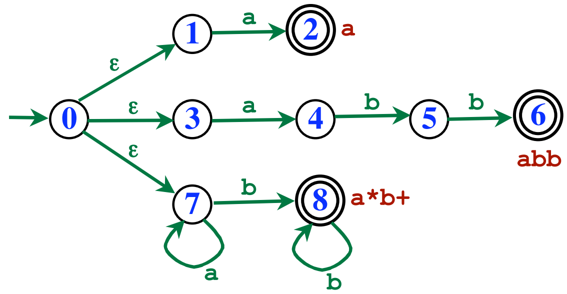
Example

Input:

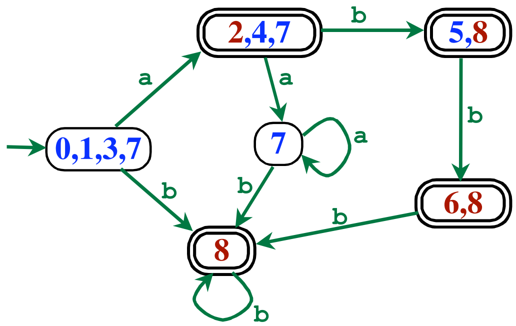
`a` { Action-1 }
`abb` { Action-2 }
`a*b+` { Action-3 }

Create NFA:

`a | abb | a*b+`



Construct Minimal DFA



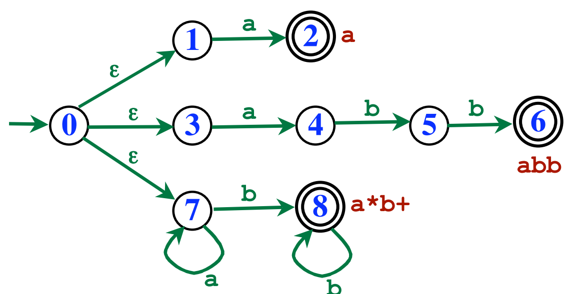
Example

Input:

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

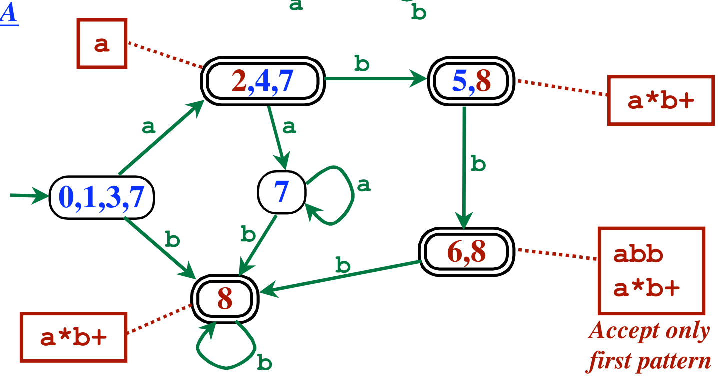
Create NFA:

`a | abb | a*b+`



Construct Minimal DFA

Attach Actions



Example

Input:

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

Create NFA:

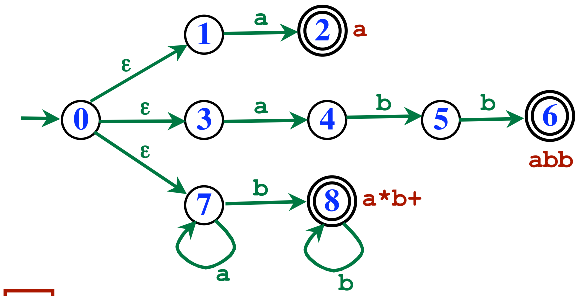
`a | abb | a*b+`

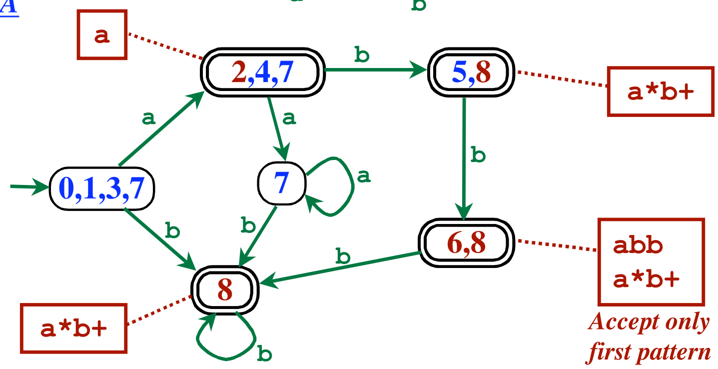
Construct Minimal DFA

Attach Actions

Example Strings:

```
a
ab
abbbbb
abb
```



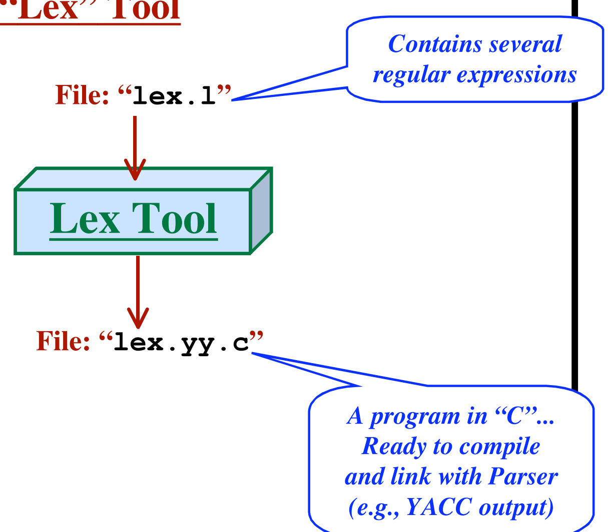


The "Lex" Tool

Oldest, most well-known
For Unix/C Environment

In UNIX:

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

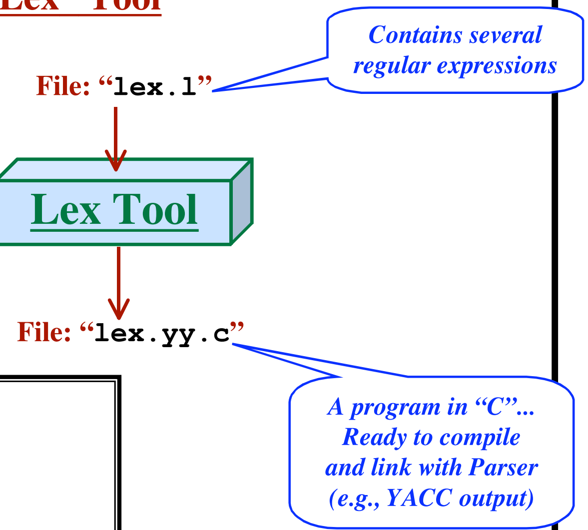


The "Lex" Tool

Oldest, most well-known
For Unix/C Environment

In UNIX:

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



Input File Format:

```
%{
        ...Any "C" Code...
}%
        ...Regular Definitions...
%%
        ...Regular Expressions with Actions...
%%
        ...Any "C" Code...
```


Regular Expressions in Lex

`abc`	Concatenation; Most characters stand for themselves

Meta Charaters:

`|`	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]`
`\x`	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`

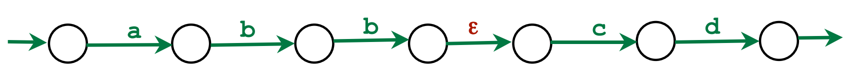
"Matches `abb`, but only if followed by `cd`."

Look-Ahead Operator, /

`abb/cd`

"Matches `abb`, but only if followed by `cd`."

Add a special ε edge for `/`

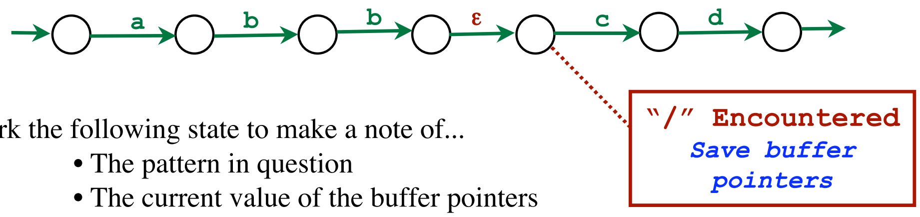


Look-Ahead Operator, /

`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.

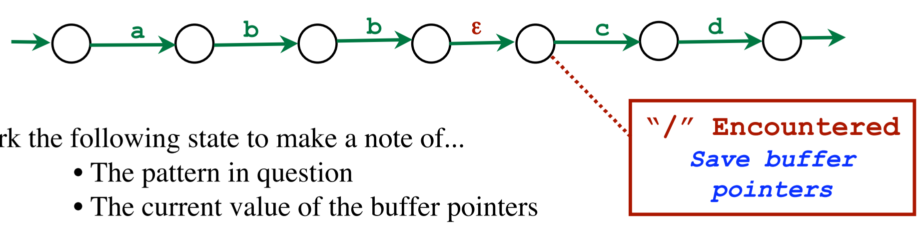
Lexical Analysis - Part 4

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...
```


Lex: Input File Format

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

Any "C" code;
Copied without changes
to beginning of the output file

Any "C" code; added to end of file
(typically, auxillary support routines)

Lex: Input File Format

```
%{
   ...Any "C" Code...
}%
   ...Regular Definitions...
   delim     [ \t\n]
   white     {delim}+
   letter    [a-zA-Z]
   digit     [0-9]
   id        {letter}({letter}|{digit})*
   num       {digit}+(\.{digit}+)?
%%
   ...Regular Expressions with Actions...
%%
   ...Any "C" Code...
```

Defined Names

Blank: Every character is Itself literally

Defined names can be used in regular expressions

Lex: Input File Format

```
%{
   ...Any "C" Code...
}%
   ...Regular Definitions...
%%
   ...Regular Expressions with Actions...
   "+"       {return PLUS;}
   "-"       {return MINUS;}
   ...
   while     {return WHILE;}
   if        {return IF;}
   ...
   {white}   {}
   ...
   {num}     {yylval = ...; return NUM;}
   {id}      {yylval = ...lookup(...)...; return ID;}
%%
   ...Any "C" Code...
```

Regular expressions

Any "C" code. Include "return" to give the token to parser.

No return means "do nothing". (This "token" is recognized but not returned to parser)

`yylval` *is where token attribute info is stored.*

You may use these variables to access the lexeme:

```
char * yytext;
int yyleng;
```