CS415 Compilers

Lexical Analysis and

These slides are based on slides copyrighted by Keith Cooper, Ken Kennedy & Linda Torczon at Rice University
Instruction Scheduling Project has been posted
Due dates: code – February 25, report – March 4
- start early
- problem is not totally defined; you may run into situations where you will need time to “redesign” your solution
- the report is 40% of the grade, code is 60%
- need to follow calling conventions since we will use automatic testing framework
- we mainly provide help for C/C++ implementations
- not a group project!
→ The scanner is the first stage in the front end
→ Specifications can be expressed using regular expressions
→ Build tables and code from a DFA
Review: Goal

- We show how to construct a finite state automaton to recognize any RE
- Overview:
  - Direct construction of a nondeterministic finite automaton (NFA) to recognize a given RE
    - Requires $\varepsilon$-transitions to combine regular subexpressions
  - Construct a deterministic finite automaton (DFA) to simulate the NFA
    - Use a set-of-states construction
  - Minimize the number of states
    - Hopcroft state minimization algorithm
  - Generate the scanner code
    - Additional specifications needed for details
Non-deterministic Finite Automata

Each RE corresponds to a deterministic finite automaton (DFA)
- May be hard to directly construct the right DFA

What about an RE such as \((a \mid b)^* abb\) ?

This is a little different
- \(S_0\) has a transition on \(\varepsilon\)
- \(S_1\) has two transitions on \(a\)

This is a non-deterministic finite automaton (NFA)
Non-deterministic Finite Automata

- An NFA accepts a string $x$ iff $\exists$ a path through the transition graph from $s_0$ to a final state such that the edge labels spell $x$
- Transitions on $\epsilon$ consume no input
- To “run” the NFA, start in $s_0$ and guess the right transition at each step
  \- Always guess correctly
  \- If some sequence of correct guesses accepts $x$ then accept

Why study NFAs?
- They are the key to automate the RE→DFA construction
- We can paste together NFAs with $\epsilon$-transitions

\[ \text{NFA} \xrightarrow{\epsilon} \text{NFA} \quad \text{becomes an} \quad \text{NFA} \]
DFA is a special case of an NFA
- DFA has no $\varepsilon$ transitions
- DFA’s transition function is single-valued
- Same rules will work

DFA can be simulated with an NFA
- Obviously

NFA can be simulated with a DFA
- Simulate sets of possible states
- Possible exponential blowup in the state space
- Still, one state per character in the input stream
Automating Scanner Construction

To convert a specification into code:
1. Write down the RE for the input language
2. Build a big NFA
3. Build the DFA that simulates the NFA
4. Systematically shrink the DFA
5. Turn it into code

Scanner generators
- Lex and Flex work along these lines
- Algorithms are well-known and well-understood
- Key issue is interface to parser (define all parts of speech)
- You could build one in a weekend!
Review: Automating Scanner Construction

RE $\rightarrow$ NFA \textit{(Thompson's construction)}

- Build an NFA for each term
- Combine them with $\varepsilon$-moves

NFA $\rightarrow$ DFA \textit{(subset construction)}

- Build the simulation

DFA $\rightarrow$ Minimal DFA

- Hopcroft's algorithm

DFA $\rightarrow$ RE \textit{(Not part of the scanner construction)}

- All pairs, all paths problem
- Take the union of all paths from $s_0$ to an accepting state
Key idea

- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with $\varepsilon$ moves in precedence order

NFA for $a$

NFA for $ab$
Key idea

- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with $\varepsilon$ moves in precedence order

NFA for $a$

NFA for $ab$
Key idea

- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with $\varepsilon$ moves in precedence order

NFA for $a$

NFA for $ab$

NFA for $a \mid b$
Key idea

- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with $\varepsilon$ moves in precedence order
Key idea

- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with $\varepsilon$ moves in precedence order

NFA for $a$

NFA for $ab$

NFA for $a \mid b$
Key idea

- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with $\varepsilon$ moves in precedence order

NFA for $a$

NFA for $ab$

NFA for $a \mid b$

NFA for $a^*$
Key idea

- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with ε moves in precedence order
Key idea

- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with $\varepsilon$ moves in precedence order

```
NFA for a
S0 → a → S1

NFA for ab
S0 → a → S1 → ε → S3 → b → S4

NFA for a ∨ b
S0 → ε → S1 → a → S2 → ε → S5
S0 → ε → S3 → b → S4
S3 → b → S4

NFA for a*
S0 → ε
S1 → a → S3
```
Key idea
- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with \( \varepsilon \) moves in precedence order

NFA for \( a \)

NFA for \( ab \)

NFA for \( a \mid b \)

NFA for \( a^* \)
Key idea

- NFA pattern for each symbol and each operator
- Each NFA has a single start and accept state
- Join them with \( \epsilon \) moves in precedence order

Ken Thompson, CACM, 1968
Example of Thompson’s Construction

Let’s try \( a \ (b \mid c)^* \)

1. \( a, b, \) & \( c \)

2. \( b \mid c \)

3. \( (b \mid c)^* \)
Let’s try $a \ (b \mid c)^*$

1. $a, \ b, \ & \ c$

2. $b \mid c$

3. $(b \mid c)^*$
4. \( a ( b | c )^* \)

Of course, a human would design something simpler ...

But, we can automate production of the more complex one ...
RE → NFA *(Thompson’s construction)*
- Build an NFA for each term
- Combine them with ε-moves

NFA → DFA *(subset construction)*
- Build the simulation

DFA → Minimal DFA
- Hopcroft’s algorithm

DFA → RE *(Not part of the scanner construction)*
- All pairs, all paths problem
- Take the union of all paths from $s_0$ to an accepting state
Need to build a simulation of the NFA

Two key functions

- \( \text{Move}(s_i, a) \) is set of states reachable from \( s_i \) by \( a \)
- \( \varepsilon\text{-closure}(s_i) \) is set of states reachable from \( s_i \) by \( \varepsilon \)

The algorithm:

- Start state derived from \( s_0 \) of the NFA
- Take its \( \varepsilon \)-closure \( S_0 = \varepsilon\text{-closure}(s_0) \)
- Take the image of \( S_0 \), \( \text{move}(S_0, a) \) for each \( a \in \Sigma \), and take its \( \varepsilon \)-closure, add it to the state set \( S \)
- For each state \( S \), compute \( \text{move}(S, a) \) for each \( a \in \Sigma \), and take its \( \varepsilon \)-closure
- Iterate until no more states are added

*Sounds more complex than it is...*
The algorithm:

\[ s_0 \leftarrow \varepsilon\text{-closure}(q_0) \]

add \( s_0 \) to \( S \)

while ( \( S \) is still changing )

for each \( s_i \in S \)

for each \( a \in \Sigma \)

\[ s_? \leftarrow \varepsilon\text{-closure}(\text{move}(s_i, a)) \]

if ( \( s_? \notin S \) ) then

add \( s_? \) to \( S \) as \( s_j \)

\[ T[ s_i, a ] \leftarrow s_j \]

e else

\[ T[ s_i, a ] \leftarrow s_? \]

Let’s think about why this works
The algorithm:

\[ s_0 \leftarrow \varepsilon\text{-closure}(q_0) \]
\[ \text{add } s_0 \text{ to } S \]
\[ \text{while ( } S \text{ is still changing ) } \]
\[ \text{for each } s_i \in S \]
\[ \text{for each } a \in \Sigma \]
\[ s_? \leftarrow \varepsilon\text{-closure}(\text{move}(s_i, a)) \]
\[ \text{if ( } s_? \notin S \text{ ) then } \]
\[ \text{add } s_? \text{ to } S \text{ as } s_j \]
\[ T[s_i, a] \leftarrow s_j \]
\[ \text{else } \]
\[ T[s_i, a] \leftarrow s_? \]

Let’s think about why this works

The algorithm halts:

1. \( S \) contains no duplicates (test before adding)
2. \( 2^Q \) is finite
3. while loop adds to \( S \), but does not remove from \( S \) (monotone)
   \[ \Rightarrow \text{the loop halts} \]

\( S \) contains all the reachable NFA states

\text{It tries each symbol in each } s_i.
\text{It builds every possible NFA configuration.}
\[ \Rightarrow S \text{ and } T \text{ form the DFA} \]
Example of a fixed-point computation
- Monotone construction of some finite set
- Halts when it stops adding to the set
- Proofs of halting & correctness are similar
- These computations arise in many contexts

Other fixed-point computations
- Canonical construction of sets of LR(1) items
  → Quite similar to the subset construction
- Classic data-flow analysis
  → Solving sets of simultaneous set equations
- DFA minimization algorithm (coming up!)

We will see many more fixed-point computations
NFA → DFA with Subset Construction

\[ a ( b | c )^* : \]

Applying the subset construction:

<table>
<thead>
<tr>
<th>( s_0 )</th>
<th>( q_0 )</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \varepsilon \)-closure \( \text{move}(s, \ast) \)
$a (b \mid c)^*$:

![Diagram](image)

Applying the subset construction:

<table>
<thead>
<tr>
<th>NFA states</th>
<th>$\varepsilon$-closure(move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$q_0$</td>
</tr>
<tr>
<td></td>
<td>$q_1, q_2, q_3, q_4, q_6, q_9$</td>
</tr>
<tr>
<td>$q_0$</td>
<td>$q_1, q_2, q_3, q_4, q_6, q_9$</td>
</tr>
<tr>
<td></td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>none</td>
</tr>
</tbody>
</table>
a (b | c)*:

Applying the subset construction:

<table>
<thead>
<tr>
<th>NFA states</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 )</td>
<td>( q_0 )</td>
<td>( q_1, q_2, q_3, q_4, q_6, q_9 )</td>
<td>none</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>( q_1, q_2, q_3, q_4, q_6, q_9 )</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>
a (b | c)*:

Applying the subset construction:

<table>
<thead>
<tr>
<th>NFA states</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_0) (q_0)</td>
<td>(q_1, q_2, q_3, q_4, q_6, q_9)</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>(s_1) (q_1, q_2, q_3, q_4, q_6, q_9)</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>
\( (b | c)^* \):

Applying the subset construction:

<table>
<thead>
<tr>
<th>NFA states</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 )</td>
<td>( q_0  )</td>
<td>( q_1, q_2, q_3, q_4, q_6, q_9 )</td>
<td>none</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>( q_1, q_2, q_3, q_4, q_6, q_9 )</td>
<td>none</td>
<td>( q_5 )</td>
</tr>
</tbody>
</table>
Applying the subset construction:

<table>
<thead>
<tr>
<th></th>
<th>NFA states</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$q_0$</td>
<td>$q_1$, $q_2$, $q_3$, $q_4$, $q_6$, $q_9$</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$q_1$, $q_2$, $q_3$, $q_4$, $q_6$, $q_9$</td>
<td>none</td>
<td>$q_5$, $q_8$, $q_9$, $q_3$, $q_4$, $q_6$</td>
<td>$q_7$, $q_8$, $q_9$, $q_3$, $q_4$, $q_6$</td>
</tr>
</tbody>
</table>
\( a \ (b \mid c)^* \):

Applying the subset construction:

<table>
<thead>
<tr>
<th>NFA states</th>
<th>( \varepsilon )-closure(move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>s_0</strong></td>
<td>( q_0 )  ( q_1, q_2, q_3, ) ( q_4, q_6, q_9 ) none none</td>
</tr>
<tr>
<td><strong>s_1</strong></td>
<td>( q_1, q_2, q_3, ) ( q_4, q_6, q_9 ) none ( q_5, q_8, q_9, ) ( q_3, q_4, q_6 )</td>
</tr>
<tr>
<td><strong>s_2</strong></td>
<td>( q_5, q_8, q_9, ) ( q_3, q_4, q_6 )</td>
</tr>
</tbody>
</table>
NFA $\rightarrow$ DFA with Subset Construction

$a (b \mid c)^*$:

Applying the subset construction:

<table>
<thead>
<tr>
<th>NFA states</th>
<th>$\epsilon$-closure(move($s, \epsilon$))</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$q_0$, $q_1$, $q_2$, $q_3$, $q_4$, $q_6$, $q_9$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$q_1$, $q_2$, $q_3$, $q_4$, $q_6$, $q_9$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$q_5$, $q_8$, $q_9$, $q_3$, $q_4$, $q_6$</td>
</tr>
</tbody>
</table>

$a$: $q_0 \rightarrow q_1 \epsilon q_2 \epsilon q_3 \epsilon q_4 \epsilon q_5 \epsilon q_6 \epsilon q_7 \epsilon q_8 \epsilon q_9$

$b$: $q_5 \rightarrow q_6 \epsilon q_7 \epsilon q_8 \epsilon q_9$

$c$: $q_5 \rightarrow q_6 \epsilon q_7 \epsilon q_8 \epsilon q_9$

$\epsilon$: $q_0 \rightarrow q_1 \epsilon q_2 \epsilon q_3 \epsilon q_4 \epsilon q_5 \epsilon q_6 \epsilon q_7 \epsilon q_8 \epsilon q_9$

$\epsilon$-closure(move($s, \epsilon$))
NFA $\rightarrow$ DFA with Subset Construction

**a (b | c)*:**

Applying the subset construction:

<table>
<thead>
<tr>
<th>NFA states</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$q_0$</td>
<td>$q_1, q_2, q_3,$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$q_4, q_6, q_9$</td>
<td></td>
</tr>
<tr>
<td>$s_1$</td>
<td>$q_1, q_2, q_3,$</td>
<td>none</td>
<td>$q_5, q_8, q_9,$</td>
</tr>
<tr>
<td></td>
<td>$q_4, q_6, q_9$</td>
<td></td>
<td>$q_3, q_4, q_6$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$q_5, q_8, q_9,$</td>
<td>none</td>
<td>$s_2$</td>
</tr>
<tr>
<td></td>
<td>$q_3, q_4, q_6$</td>
<td></td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$q_7, q_8, q_9,$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$q_3, q_4, q_6$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\( a ( b \mid c)^* : \)

Applying the subset construction:

<table>
<thead>
<tr>
<th>NFA states</th>
<th>( \epsilon )-closure(move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 )</td>
<td>( q_0 \quad q_1, q_2, q_3, )</td>
</tr>
<tr>
<td></td>
<td>( q_4, q_6, q_9 \quad none )</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>( q_1, q_2, q_3, )</td>
</tr>
<tr>
<td></td>
<td>( q_4, q_6, q_9 \quad none )</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>( q_5, q_8, q_9, )</td>
</tr>
<tr>
<td></td>
<td>( q_3, q_4, q_6 \quad none )</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>( q_7, q_8, q_9, )</td>
</tr>
<tr>
<td></td>
<td>( q_3, q_4, q_6 \quad none )</td>
</tr>
</tbody>
</table>
a (b | c)*:

\[
\begin{array}{ccc}
q_0 & \xrightarrow{a} & q_1 \\
 & & \xrightarrow{\varepsilon} q_2 \\
 & & \xrightarrow{\varepsilon} q_3 \\
& & \xrightarrow{\varepsilon} q_4 \\
& & \xrightarrow{\varepsilon} q_5 \\
& & \xrightarrow{c} q_6 \\
& & \xrightarrow{\varepsilon} q_7 \\
& & \xrightarrow{\varepsilon} q_8 \\
& & \xrightarrow{\varepsilon} q_9
\end{array}
\]

Applying the subset construction:

<table>
<thead>
<tr>
<th>NFA states</th>
<th>ε-closure(move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_0)</td>
<td>(q_0)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>(q_1, q_2, q_3, q_4, q_6, q_9)</td>
</tr>
<tr>
<td>(s_2)</td>
<td>(q_5, q_8, q_9, q_3, q_4, q_6)</td>
</tr>
<tr>
<td>(s_3)</td>
<td>(q_7, q_8, q_9, q_3, q_4, q_6)</td>
</tr>
</tbody>
</table>

Final states
The DFA for $a (b \mid c)^*$

- Ends up smaller than the NFA
- All transitions are deterministic

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_1$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$s_1$</td>
<td>-</td>
<td>$s_2$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>-</td>
<td>$s_2$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>-</td>
<td>$s_2$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Automating Scanner Construction

RE $\rightarrow$ NFA \textit{(Thompson's construction)}
- Build an NFA for each term
- Combine them with $\varepsilon$-moves

NFA $\rightarrow$ DFA \textit{(subset construction)}
- Build the simulation

DFA $\rightarrow$ Minimal DFA
- Hopcroft's algorithm

DFA $\rightarrow$ RE \textit{(not really part of scanner construction)}
- All pairs, all paths problem
- Union together paths from $s_0$ to a final state

\textit{The Cycle of Constructions}:
- RE $\rightarrow$ NFA $\rightarrow$ DFA $\rightarrow$ Minimal DFA
The Big Picture

- Discover sets of equivalent states
- Represent each such set with just one state
DFA Minimization

The Big Picture

• Discover sets of equivalent states
• Represent each such set with just one state

Two states are equivalent if and only if:

• $\forall a \in \Sigma$, transitions on $a$ lead to equivalent states (DFA)
• $a$-transitions to distinct sets $\Rightarrow$ states must be in distinct sets
The Big Picture

• Discover sets of equivalent states
• Represent each such set with just one state

Two states are equivalent if and only if:

• \( \forall a \in \Sigma, \) transitions on \( a \) lead to equivalent states \( (\text{DFA}) \)
• \( a \)-transitions to distinct sets \( \Rightarrow \) states must be in distinct sets

A partition \( P \) of \( S \)

• Each state \( s \in S \) is in exactly one set \( p_i \in P \)
• The algorithm iteratively partitions the DFA’s states
Details of the algorithm

- Group states into maximal size sets, optimistically
- Iteratively subdivide those sets, as needed
- States that remain grouped together are equivalent

Initial partition, $P_0$, has two sets: $\{F\} \& \{Q-F\}$  
($D=(Q,\Sigma,\delta, q_0, F)$)

Splitting a set (“partitioning a set by $a$”)

- Assume $q_a \& q_b \in s$, and $\delta(q_a, a) = q_x$, $\delta(q_b, a) = q_y$
- If $q_x \& q_y$ are not in the same set, then $s$ must be split
  $\rightarrow q_a$ has transition on $a$, $q_b$ does not $\Rightarrow a$ splits $s$
DFA Minimization

The algorithm

\[ P \leftarrow \{ F, \{Q-F}\} \]
while (P is still changing)
\[ T \leftarrow \{ \} \]
for each set \( S \in P \)
\[ T \leftarrow T \cup \text{split}(S) \]
\[ P \leftarrow T \]

\text{split}(S): \\
for each \( a \in \Sigma \)
if \( a \) splits \( S \) into \( S_1, S_2, \ldots \) then
return \( \{S_1, S_2, \ldots\} \)
else return \( S \)

Why does this work?

- Partition \( P \in 2^Q \)
- Start off with 2 subsets of \( Q \) \{F\} and \{Q-F\}
- While loop takes \( P_i\rightarrow P_{i+1} \) by splitting 1 or more sets
- \( P_{i+1} \) is at least one step closer to the partition with \( |Q| \) sets
- Maximum of \(|Q| \) splits

Note that
- Partitions are never combined
The algorithm

\[
P \leftarrow \{ F, \{Q-F}\} \\
\text{while (} P \text{ is still changing)} \\
\quad T \leftarrow \{\} \\
\quad \text{for each set } S \in P \\
\quad \quad T \leftarrow T \cup \text{split}(S) \\
\quad P \leftarrow T
\]

\text{split}(S):

\quad \text{for each } a \in \Sigma \\
\quad \quad \text{if } a \text{ splits } S \text{ into } S_1, S_2, \ldots \text{ then } \\
\quad \quad \quad \text{return } \{S_1, S_2, \ldots\} \\
\quad \quad \text{else return } S

Why does this work?

- Partition \( P \in 2^Q \)
- Start off with 2 subsets of \( Q \) \( \{F\} \) and \( \{Q-F\} \)
- While loop takes \( P_i \rightarrow P_{i+1} \) by splitting 1 or more sets
- \( P_{i+1} \) is at least one step closer to the partition with \( |Q| \) sets
- Maximum of \( |Q| \) splits

Note that

- Partitions are never combined

This is a fixed-point algorithm!
Then, apply the minimization algorithm

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

\[ \{s_1, s_2, s_3\} \{s_0\} \]

To produce the minimal DFA

We observed that a human would design a simpler automaton than Thompson’s construction & the subset construction did.

Minimizing that DFA produces the one that a human would design!
Start with a regular expression
\( r_0 \mid r_1 \mid r_2 \mid r_3 \mid r_4 \mid r_5 \mid r_6 \mid r_7 \mid r_8 \mid r_9 \)

The Cycle of Constructions

\[ \text{RE} \rightarrow \text{NFA} \rightarrow \text{DFA} \rightarrow \text{minimal DFA} \]
Thompson’s construction produces

The Cycle of Constructions
The subset construction builds

This is a DFA, but it has a lot of states ...

The Cycle of Constructions

RE → NFA → DFA → minimal DFA
The DFA minimization algorithm builds

This looks like what a skilled compiler writer would do!

The Cycle of Constructions
Syntax Analysis

Read EaC: 3.1 – 3.3