Lexical Analysis, II

Chapter 2 in EaC2e
Determinism (or not)

So far, we have only looked at deterministic automata, or DFAs

- **DFA** ≡ Deterministic Finite Automaton
- Deterministic means that it has only one transition out of a state on a given character

![Diagram](image)

rather than
Determinism (or not)

So far, we have only looked at deterministic automata, or DFAs

• **DFA** ≡ Deterministic Finite Automaton
• Deterministic means that it has only one transition out of a state on a given character

\[
\begin{array}{c}
 s_0 \xrightarrow{a} s_1 \\
\end{array}
\]

rather than

\[
\begin{array}{c}
 s_0 \xrightarrow{\epsilon} s_2 \\
 s_2 \xrightarrow{a} s_3 \\
\end{array}
\]

• Can a finite automaton have multiple transitions out of a single state on the same character?
  – Yes, we call such an FA a Nondeterministic Finite Automaton
  – And, yes, the NFA is truly an odd notion ... but a useful one

• **NFA**s and **DFA**s are equivalent
  – The set of DFAs is a subset of the set of NFAs
  – For any NFA, we can build a DFA that simulates its behavior

\[\Rightarrow\text{We should not worry that the } \epsilon\text{-transitions do not consume input}\]
Whoa. What Does That **NFA** “Mean”?  

An NFA accepts a string $x$ iff $\exists$ a path though the transition graph from $s_0$ to a final state such that the edge labels spell $x$, ignoring $\varepsilon$’s

Two models for NFA execution

1. To “run” the NFA, start in $s_0$ and **guess** the right transition at each step †
2. To “run” the NFA, start in $s_0$ and, at each non-deterministic choice, clone the NFA to pursue all possible paths. If any of the clones succeeds, **accept**

![NFA Diagram](image)

**NFA** for “what | who”

Note that this same operational definition works on a **DFA**

† See page 44 in EaC2e.
Why Do We Care?

We need a construction that takes an RE to a DFA to a scanner. NFAs will help us get there.

Overview:

1. Simple and direct construction of a nondeterministic finite automaton (NFA) to recognize a given RE
   - Easy to build in an algorithmic way
   - Key idea is to combine NFAs for the terms with $\varepsilon$-transitions
2. Construct a deterministic finite automaton (DFA) that simulates the NFA
   - Use a set-of-states construction
3. Minimize the number of states in the DFA
   - We will look at 2 different algorithms: Hopcroft’s & Brzozowski’s
4. Generate the scanner code
   - Additional specifications needed for the actions

lex and flex work along these lines
Example of a DFA

Here is a DFA for \(( a | b )^* \) abb

![DFA Diagram]

This DFA is not particularly obvious from the RE.

Each RE corresponds to one or more \textit{deterministic finite automata} (DFAs)

- We know a DFA exists for each RE
- The DFA may be hard to build directly
- Automatic techniques will build it for us ...

For algorithm aficionados in the class, this DFA is reminiscent of the way that the failure function works in the Knuth, Morris, & Pratt sub-linear time pattern matcher.
Example as an NFA

Here is a simpler, more obvious NFA for \((a \mid b)^* abb\)

Here is an NFA for the same language

• The relationship between the RE and the NFA is more obvious
• The \(\varepsilon\)-transition pastes together two DFAs to form a single NFA
• We can rewrite this NFA to eliminate the \(\varepsilon\)-transition
  – \(\varepsilon\)-transitions are an odd and convenient quirk of NFAs
  – Eliminating this one makes it obvious that it has 2 transitions on \(a\) from \(s_0\)
Relationship between NFAs and DFAs

**DFA** is a special case of an **NFA**

- **DFA** has no ε 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

⇒ **NFA** & **DFA** are equivalent in ability to recognize languages

*Rabin & Scott, 1959*
The Plan for Scanner Construction

**RE → NFA** *(Thompson’s construction)*
- Build an **NFA** for each term in the **RE**
- Combine them in patterns that model the operators

**NFA → DFA** *(Subset construction)*
- Build a **DFA** that simulates the **NFA**

**DFA → Minimal DFA**
- Hopcroft’s algorithm
- Brzozowski’s algorithm

**Minimal DFA → Scanner**
- See §2.5 in EaC2e

**DFA → RE**
- All pairs, all paths problem
- Union together paths from $s_0$ to a final state

**Automata Theory Moment**
Taken together, the constructions in the cycle show that **REs**, **NFA**s, and **DFA**s are all equivalent in their expressive power.

**The Cycle of Constructions**

*Taken together, these constructions prove that DFAs and REs are equivalent.*
**RE → NFA using Thompson’s Construction**

**Key idea**
- For each symbol & each operator, we have an NFA pattern
- Join them with ε moves in precedence order

**NFA for a**

**NFA for ab**

**NFA for a | b**

**NFA for a**

**Precedence in REs:**
- Closure
- Concatenation
- Alternation

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

Let’s build an NFA for \( a (b \mid c)^* \)

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

\[
\begin{array}{c}
S_0 \xrightarrow{a} S_1 \\
S_0 \xrightarrow{b} S_1 \\
S_0 \xrightarrow{c} S_1 \\
\end{array}
\]

2. \( b \mid c \)

\[
\begin{array}{c}
S_0 \xrightarrow{\varepsilon} S_1 \\
S_1 \xrightarrow{b} S_2 \\
S_2 \xrightarrow{\varepsilon} S_5 \\
S_3 \xrightarrow{\varepsilon} S_4 \\
S_4 \xrightarrow{c} S_3 \\
S_5 \xrightarrow{\varepsilon} S_5 \\
\end{array}
\]

3. \( (b \mid c)^* \)

\[
\begin{array}{c}
S_0 \xrightarrow{\varepsilon} S_1 \\
S_1 \xrightarrow{\varepsilon} S_2 \\
S_2 \xrightarrow{b} S_3 \\
S_3 \xrightarrow{\varepsilon} S_6 \\
S_4 \xrightarrow{\varepsilon} S_5 \\
S_5 \xrightarrow{\varepsilon} S_5 \\
S_6 \xrightarrow{\varepsilon} S_7 \\
\end{array}
\]

Note that states are being renamed at each step.
Example of Thompson’s Construction

4.  \( a^* (b \mid c)^* \)

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

But, we can automate production of the more complex NFA version ...

COMP 412, Fall 2017
Thompson’s Construction

Warning

• You will be tempted to take shortcuts, such as leaving out some of the $\varepsilon$ transitions
• Do not do it. Memorize these four patterns. They will keep you out of trouble.
The Plan for Scanner Construction

\[ \text{RE} \rightarrow \text{NFA} \quad (\text{Thompson’s construction}) \]
- Build an \text{NFA} for each term in the \text{RE}
- Combine them in patterns that model the operators

\[ \text{NFA} \rightarrow \text{DFA} \quad (\text{Subset construction}) \]
- Build a \text{DFA} that simulates the \text{NFA}

\[ \text{DFA} \rightarrow \text{Minimal DFA} \]
- Hopcroft’s algorithm
- Brzozowski’s algorithm

\[ \text{Minimal DFA} \rightarrow \text{Scanner} \]
- See § 2.5 in EaC2e

\[ \text{DFA} \rightarrow \text{RE} \]
- All pairs, all paths problem
- Union together paths from \( s_0 \) to a final state
Simulating an **NFA** with a **DFA**

Where the mapping between **NFA** states and **DFA** states is:

<table>
<thead>
<tr>
<th>DFA</th>
<th>NFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_0$</td>
<td>$n_0$</td>
</tr>
<tr>
<td>$d_1$</td>
<td>$n_1$ $n_2$ $n_3$ $n_4$ $n_6$ $n_9$</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$n_5$ $n_8$ $n_9$ $n_3$ $n_4$ $n_6$</td>
</tr>
<tr>
<td>$d_3$</td>
<td>$n_7$ $n_8$ $n_9$ $n_3$ $n_4$ $n_6$</td>
</tr>
</tbody>
</table>
NFA → DFA with Subset Construction

The subset construction builds a DFA that simulates the NFA

Two key functions

• Move\((s_i, a)\) is the set of states reachable from \(s_i\) by \(a\)

• FollowEpsilon\((s_i)\) is the set of states reachable from \(s_i\) by \(\varepsilon\)

The algorithm

• Derive the DFA’s start state from \(n_0\) of the NFA

• Add all states reachable from \(n_0\) by following \(\varepsilon\)
  – \(d_0 = \text{FollowEpsilon}\(\{n_0\}\)\)
  – Let \(D = \{d_0\}\)

• For \(\alpha \in \Sigma\), compute \(\text{FollowEpsilon}\(\text{Move}(d_0, \alpha)\)\)
  – If this creates a new state, add it to \(D\)

• Iterate until no more states are added

It sounds more complex than it is...
The algorithm:

d_0 \leftarrow \text{FollowEpsilon}(\{ n_0 \})
D \leftarrow \{ d_0 \}
W \leftarrow \{ d_0 \}
while (W \neq \emptyset) {
    select and remove s from W
    for each \( \alpha \in \Sigma \) {
        t \leftarrow \text{FollowEpsilon}(\text{Move}(s, \alpha))
        T[s, \alpha] \leftarrow t
        if (t \notin D) then {
            add t to D
            add t to W
        }
    }
}

This test is a little tricky

d_0 is a set of states
D & W are sets of sets of states

The algorithm halts:
1. \( D \) contains no duplicates (test before addition)
2. \( 2^{\text{NFA states}} \) is finite
3. while loop adds to \( D \), but does not remove from \( D \) (monotone)
\( \Rightarrow \) the loop halts

\( D \) contains all the reachable NFA states

It tries each character in each \( d_i \).

It builds every possible NFA configuration.

\( \Rightarrow D \) and \( T \) form the DFA

Any DFA state that contains a final state of the NFA becomes a final state of the DFA.
NFA $\rightarrow$ DFA with Subset Construction

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 & Gaussian Elimination
  - Solving sets of simultaneous set equations

We will see many more fixed-point computations
NFA $\rightarrow$ DFA with Subset Construction

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

\[
\begin{array}{c}
\text{States} \\
\text{DFA} & \text{NFA} & a & b & c \\
\hline
d_0 & n_0 & \\
\end{array}
\]

\[
\begin{array}{c}
\text{FollowEpsilon (Move(s,*))} \\
\end{array}
\]
\[ a (b \mid c)^* : \]

\[
\begin{array}{c}
\text{States} \\
\text{DFA} & \text{NFA} \\
\hline
d_0 & n_0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
& \text{FollowEpsilon ( Move( s, * ) )} \\
\text{a} & n_1, n_2, n_3 & n_4, n_6, n_9 \\
\text{b} & & \\
\text{c} & n_1, n_2, n_3 & n_4, n_6, n_9 \\
\end{array}
\]
**NFA → DFA with Subset Construction**

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

![Diagram of NFA and DFA transition](image)

<table>
<thead>
<tr>
<th>States</th>
<th>FollowEpsilon (Move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>NFA</td>
</tr>
<tr>
<td>(d_0)</td>
<td>(n_0)</td>
</tr>
</tbody>
</table>

- For a: \(n_1 n_2 n_3 n_4 n_6 n_9\)
- For b: none
- For c: none

COMP 412, Fall 2017
**NFA → DFA with Subset Construction**

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

![Diagram of NFA to DFA conversion]

<table>
<thead>
<tr>
<th>States</th>
<th>FollowEpsilon ( Move( s, * ) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>NFA</td>
</tr>
<tr>
<td>( d_0 )</td>
<td>( n_0 )</td>
</tr>
<tr>
<td>( d_1 )</td>
<td>( n_1 \ n_2 \ n_3 \ n_4 \ n_6 \ n_9 )</td>
</tr>
</tbody>
</table>
NFA $\rightarrow$ DFA with Subset Construction

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

![NFA to DFA Diagram](image)

<table>
<thead>
<tr>
<th>States</th>
<th>FollowEpsilon ( Move( s, * ) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>NFA</td>
</tr>
<tr>
<td>$d_0$</td>
<td>$n_0$</td>
</tr>
<tr>
<td>$d_1$</td>
<td>$n_1$ $n_2$ $n_3$</td>
</tr>
<tr>
<td></td>
<td>$n_4$ $n_6$ $n_9$</td>
</tr>
</tbody>
</table>

COMP 412, Fall 2017
NFA $\rightarrow$ DFA with Subset Construction

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

```plaintext
States

<table>
<thead>
<tr>
<th>DFA</th>
<th>NFA</th>
<th>FollowEpsilon ( Move( s, * ) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>$d_0$</td>
<td>$n_0$</td>
<td>${ n_1, n_3 }$</td>
</tr>
<tr>
<td>$d_1$</td>
<td>${ n_1, n_2, n_3, n_4, n_6, n_9 }$</td>
<td>${ n_1, n_2, n_3, n_4, n_6, n_9 }$</td>
</tr>
</tbody>
</table>
```

COMP 412, Fall 2017
## NFA → DFA with Subset Construction

### States

<table>
<thead>
<tr>
<th>DFA</th>
<th>NFA</th>
<th>FollowEpsilon ( Move(s,∗) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_0$</td>
<td>$n_0$</td>
<td>$n_1$ $n_2$ $n_3$ $n_4$ $n_6$ $n_9$</td>
</tr>
<tr>
<td>$d_1$</td>
<td>$n_1$ $n_2$ $n_3$ $n_4$ $n_6$ $n_9$</td>
<td>none</td>
</tr>
</tbody>
</table>
NFA → DFA with Subset Construction

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

States

<table>
<thead>
<tr>
<th>DFA</th>
<th>NFA</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_0 )</td>
<td>( n_0 )</td>
<td>( n_1 \ n_2 \ n_3 \n_4 \ n_6 \ n_9 )</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>( d_1 )</td>
<td>( n_1 \ n_2 \ n_3 \n_4 \ n_6 \ n_9 )</td>
<td>none</td>
<td>( n_5 \ n_8 \ n_9 \n_3 \ n_4 \ n_6 )</td>
<td>( n_7 \ n_8 \ n_9 \n_3 \ n_4 \ n_6 )</td>
</tr>
<tr>
<td>( d_2 )</td>
<td>( n_5 \ n_8 \ n_9 \n_3 \ n_4 \ n_6 )</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

FollowEpsilon ( Move( s, * ) )
NFA $\rightarrow$ DFA with Subset Construction

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

![Diagram](image)

<table>
<thead>
<tr>
<th>States</th>
<th>FollowEpsilon (Move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>NFA</td>
</tr>
<tr>
<td>d₀</td>
<td>n₀</td>
</tr>
<tr>
<td>d₁</td>
<td>n₁ n₂ n₃ n₄ n₆ n₉</td>
</tr>
<tr>
<td>d₂</td>
<td>n₅ n₈ n₉ n₃ n₄ n₆</td>
</tr>
<tr>
<td>d₃</td>
<td>n₇ n₈ n₉ n₃ n₄ n₆</td>
</tr>
</tbody>
</table>

COMP 412, Fall 2017
**NFA → DFA with Subset Construction**

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

![Graph showing transitions between states](image)

<table>
<thead>
<tr>
<th>States</th>
<th>FollowEpsilon (Move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>NFA</td>
</tr>
<tr>
<td>$d_0$</td>
<td>$n_0$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>$d_1$</td>
<td>$n_1 \ n_2 \ n_3$</td>
</tr>
<tr>
<td></td>
<td>$n_4 \ n_6 \ n_9$</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$n_5 \ n_8 \ n_9$</td>
</tr>
<tr>
<td></td>
<td>$n_3 \ n_4 \ n_6$</td>
</tr>
<tr>
<td>$d_3$</td>
<td>$n_7 \ n_8 \ n_9$</td>
</tr>
<tr>
<td></td>
<td>$n_3 \ n_4 \ n_6$</td>
</tr>
</tbody>
</table>
**NFA → DFA with Subset Construction**

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

- **NFA**:
  - States:
    - \(n_0\)
    - \(n_1\)
    - \(n_2\)
    - \(n_3\)
    - \(n_4\)
    - \(n_5\)
    - \(n_6\)
    - \(n_7\)
    - \(n_8\)
    - \(n_9\)
  - Transitions:
    - \(a\) from \(n_0\) to \(n_1\)
    - \(\varepsilon\) from \(n_1\) to \(n_2\), \(n_3\)
    - \(\varepsilon\) from \(n_2\) to \(n_3\), \(n_4\)
    - \(b\) from \(n_3\) to \(n_5\)
    - \(\varepsilon\) from \(n_4\) to \(n_6\), \(n_7\)
    - \(\varepsilon\) from \(n_5\) to \(n_8\)
    - \(\varepsilon\) from \(n_7\) to \(n_8\), \(n_9\)

- **DFA**:
  - States:
    - \(d_0\)
    - \(d_1\)
    - \(d_2\)
    - \(d_3\)
  - Transitions:
    - \(a\) from \(d_1\) to \(n_1\), \(n_2\), \(n_3\), \(n_4\), \(n_5\), \(n_7\), \(n_8\), \(n_9\)
    - \(b\) from \(d_2\) to \(n_5\), \(n_8\), \(n_9\)
    - \(c\) from \(d_3\) to \(n_7\), \(n_8\), \(n_9\)

**States**

<table>
<thead>
<tr>
<th>DFA</th>
<th>NFA</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_0)</td>
<td>(n_0)</td>
<td>(n_1)</td>
<td>(n_2)</td>
<td>(n_3)</td>
</tr>
<tr>
<td>(d_1)</td>
<td>(n_1), (n_2), (n_3), (n_4), (n_5), (n_7), (n_8), (n_9)</td>
<td>none</td>
<td>(n_5), (n_8), (n_9)</td>
<td>(n_7), (n_8), (n_9)</td>
</tr>
<tr>
<td>(d_2)</td>
<td>(n_5), (n_8), (n_9), (n_3), (n_4), (n_6)</td>
<td>none</td>
<td>(d_2)</td>
<td>none</td>
</tr>
<tr>
<td>(d_3)</td>
<td>(n_7), (n_8), (n_9), (n_3), (n_4), (n_6)</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

\(n_7\) is the core state of \(d_3\)
a ( b | c )* :

DFA

NFA

<table>
<thead>
<tr>
<th>States</th>
<th>DFA</th>
<th>NFA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d_0$</td>
<td>$n_0$</td>
</tr>
<tr>
<td>$d_1$</td>
<td>$n_1$ $n_2$ $n_3$ $n_4$ $n_6$ $n_9$</td>
<td></td>
</tr>
<tr>
<td>$d_2$</td>
<td>$n_5$ $n_8$ $n_9$ $n_3$ $n_4$ $n_6$</td>
<td></td>
</tr>
<tr>
<td>$d_3$</td>
<td>$n_7$ $n_8$ $n_9$ $n_3$ $n_4$ $n_6$</td>
<td></td>
</tr>
</tbody>
</table>

FollowEpsilon ( Move( $s$, $*$ ) )

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n_1$ $n_2$ $n_3$ $n_4$ $n_6$ $n_9$</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>$d_1$</td>
<td>none</td>
<td>$n_5$ $n_8$ $n_9$</td>
<td>$n_7$ $n_8$ $n_9$</td>
</tr>
<tr>
<td>$d_2$</td>
<td>none</td>
<td>$d_2$</td>
<td>$d_3$</td>
</tr>
<tr>
<td>$d_3$</td>
<td>none</td>
<td>$d_2$</td>
<td>$d_3$</td>
</tr>
</tbody>
</table>

$n_5$ is the core state of $d_2$
### NFA → DFA with Subset Construction

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

![NFA DFA Subset Construction Diagram](image)

<table>
<thead>
<tr>
<th>States</th>
<th>FollowEpsilon (Move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>NFA</td>
</tr>
<tr>
<td>d₀</td>
<td>n₀</td>
</tr>
<tr>
<td>d₁</td>
<td>n₁ n₂ n₃ n₄ n₆ n₉</td>
</tr>
<tr>
<td>d₂</td>
<td>n₅ n₈ n₉ n₃ n₄ n₆</td>
</tr>
<tr>
<td>d₃</td>
<td>n₇ n₈ n₉ n₃ n₄ n₆</td>
</tr>
</tbody>
</table>

Final states because of n₉
NFA $\rightarrow$ DFA with Subset Construction

$\mathbf{a} \ (\mathbf{b} \ | \ \mathbf{c})^*$:

![Transition Diagram]

<table>
<thead>
<tr>
<th>States</th>
<th>FollowEpsilon (Move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DFA</td>
</tr>
<tr>
<td>$d_0$</td>
<td>$n_0$</td>
</tr>
<tr>
<td>$d_1$</td>
<td>$n_1 \ n_2 \ n_3$</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$n_5 \ n_8 \ n_9$</td>
</tr>
<tr>
<td>$d_3$</td>
<td>$n_7 \ n_8 \ n_9$</td>
</tr>
</tbody>
</table>

Transition table for the DFA
The DFA for \( a (b \mid c)^* \)

- Much smaller than the NFA (no \( \epsilon \)-transitions)
- All transitions are deterministic
- Use same code skeleton as before

But, remember, our goal was:
chines are more general than the ordinary ones, but this is not the case. We shall give a direct construction of an ordinary automaton, defining exactly the same set of tapes as a given nondeterministic machine.

**Definition 11.** Let \( A = (S,M,S_0,F) \) be a nondeterministic automaton. \( \mathcal{D}(A) \) is the system \((T,N,t_0,G)\) where \( T \) is the set of all subsets of \( S \), \( N \) is a function on \( T \times \Sigma \) such that \( N(t,o) \) is the union of the sets \( M(s,o) \) for \( s \) in \( t \), \( N(t_0, \cdot) = t_0 = S_0 \), and \( G \) is the set of all subsets of \( S \) containing at least one member of \( T \). Clearly \( \mathcal{D}(A) \) is an ordinary automaton, but it is actually equivalent to \( A \).

**Theorem 11.** If \( A \) is a nondeterministic automaton, then \( T(A) = T(\mathcal{D}(A)) \).

**Proof:** Assume first that \( T(A) \) and \( \mathcal{D}(A) \) have states satisfying the conditions of Definition 10. We show by induction that for \( k=n \), \( s_k \) is in \( N(t_0,t_k) \). For \( k=0 \), \( N(t_0,t_0) = N(t_0) = t_0 = S_0 \), and we were given that \( S_0 \) is in \( S_0 \). Assume the result for \( k-1 \). By definition, \( N(t_0,t_k) = N(N(t_0,t_{k-1}),s_{k-1}) \). We have assumed \( s_{k-1} \) is in \( N(t_0,t_{k-1}) \), so that from the definition of \( N \) we have \( M(s_{k-1},s_{k-1}) \subseteq N(t_0,t_k) \). However, \( s_k \) is in \( M(s_{k-1},s_{k-1}) \), and so the result is established. In particular, \( s_n \) is in \( N(t_0,t_n) = N(t_0,t) \), and since \( s_n \) is in \( F \), we have \( N(t_0,t) \) in \( G \), which proves that \( x \) is in \( T(\mathcal{D}(A)) \). Hence, we have shown that \( T(A) \subseteq T(\mathcal{D}(A)) \).

Assume next that a tape \( x = \sigma_0\sigma_1\ldots\sigma_{n-1} \) is in \( T(\mathcal{D}(A)) \). Let for each \( k\leq n \), \( t_k = N(t_0,t_k) \). We shall work backwards. First, we know that \( t_{n-1} \) is in \( G \). Let then \( s_n \) be any internal state of \( A \) such that \( s_n \) is in \( t_n \) and \( s_n \) is in \( F \). Since \( s_n \) is in \( t_n \), \( s_n = N(t_0,t_n) = N(t_{n-1},s_{n-1}) \), we have from the definition of \( N \) that \( s_n \) is in \( M(s_{n-1},s_{n-1}) \) for some \( s_{n-1} \) in \( t_{n-1} \). But then \( \ldots \)
The Plan for Scanner Construction

**RE → NFA (Thompson’s construction)**
- Build an NFA for each term in the RE
- Combine them in patterns that model the operators

**NFA → DFA (Subset construction)**
- Build a DFA that simulates the NFA

**DFA → Minimal DFA**
- Hopcroft’s algorithm
- Brzozowski’s algorithm

**Minimal DFA → Scanner**
- See § 2.5 in EaC2e

**DFA → RE**
- All pairs, all paths problem
- Union together paths from $s_0$ to a final state
Brzozowski’s Algorithm for DFA Minimization

The Intuition

- The subset construction merges prefixes in the NFA

Thompson’s construction would leave ε-transitions between each single-character automaton

Subset construction eliminates ε-transitions and merges the paths for a. It leaves duplicate tails, such as bc, intact.
Brzozowski’s Algorithm

Idea: Use The Subset Construction Twice

• For an NFA $N$
  – Let $\text{reverse}(N)$ be the NFA constructed by making initial state final, adding a new start state with an $\varepsilon$-transition to each previously final state, and reversing the other edges
  – Let $\text{subset}(N)$ be the DFA produced by the subset construction on $N$
  – Let $\text{reachable}(N)$ be $N$ after removing any states that are not reachable from the initial state
• Then,

$$\text{reachable}(\text{subset}(\text{reverse( reachable( subset(reverse(N) ) ) )}))$$

is a minimal DFA that implements $N$  [Brzozowski, 1962]

Not everyone finds this result to be intuitive.
Neither algorithm dominates the other.
Brzozowski’s Algorithm

Step 1

- The subset construction on \textit{reverse}(NFA) merges suffixes in original NFA
Brzozowski’s Algorithm

Step 2

• Reverse it again & use subset to merge prefixes ...

Reverse it, again

And subset it, again

The Cycle of Constructions

Minimal DFA

COMP 412, Fall 2017
Abbreviated Register Specification

Start with a regular expression

r0 | r1 | r2 | r3 | r4 | r5 | r6 | r7 | r8 | r9

Register names from zero to nine

The Cycle of Constructions
Abbreviated Register Specification

Thompson’s construction produces something along these lines:

To make the example fit, we have eliminated some of the ε-transitions, e.g., between \( r \) and \( 0 \).
Abbreviated Register Specification

Applying the subset construction yields

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

The Cycle of Constructions
Abbreviated Register Specification

Applying Brzozowski’s algorithm, step 1

Technically, this edge shows up as 10 edges, which need to be combined...

The Cycle of Constructions

COMP 412, Fall 2017
Abbreviated Register Specification

Brzozowski, step 2 reverses that DFA and subsets it again

A skilled human might build this DFA

The Critical Point:
- The construction will build a minimal DFA
- The size of the DFA relates to the language described by the RE, not the size of the RE
- The result is a DFA, so it has $O(1)$ cost per character
- The compiler writer can use the “most natural” or “most intuitive” RE
Where are we? Why are we doing this?

**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
- Brzozowski’s algorithm ✓

**DFA → RE**
- All pairs, all paths problem
- Union together paths from $s_0$ to a final state

*The Cycle of Constructions*
Kleene’s Construction

\[
\begin{align*}
&\text{for } i \leftarrow 0 \text{ to } |D| - 1; \quad \text{// label each immediate path} \\
&\quad \text{for } j \leftarrow 0 \text{ to } |D| - 1; \\
&\quad \quad R^0_{ij} \leftarrow \{ a \mid \delta(d_i, a) = d_j \}; \\
&\quad \quad \text{if } (i = j) \text{ then} \\
&\quad \quad \quad R^0_{ii} = R^0_{ii} \cup \{ \varepsilon \}; \\
&\text{for } k \leftarrow 0 \text{ to } |D| - 1; \quad \text{// label nontrivial paths} \\
&\quad \text{for } i \leftarrow 0 \text{ to } |D| - 1; \\
&\quad \quad \text{for } j \leftarrow 0 \text{ to } |D| - 1; \\
&\quad \quad \quad R^k_{ij} \leftarrow R^{k-1}_{ik} (R^{k-1}_{kk})^* R^{k-1}_{kj} \cup R^{k-1}_{ij} \\
&L \leftarrow \{ \} \quad \text{// union labels of paths from} \\
&\text{For each final state } s_i \quad \text{// } s_0 \text{ to a final state } s_i \\
&L \leftarrow L \cup R^{\lfloor D \rfloor - 1}_{0i}
\end{align*}
\]

The Wikipedia page on “Kleene’s algorithm” is pretty good. It also contains a link to Kleene’s 1956 paper. This form of the algorithm is usually attributed to McNaughton and Yamada in 1960.

Adaptation of all points, all paths, low cost algorithm

COMP 412, Fall 2017