Lexical Analysis, II

Comp 412

Copyright 2019, Keith D. Cooper & Linda Torczon, all rights reserved.

Students enrolled in Comp 412 at Rice University have explicit permission to make copies of these materials for their personal use.

Faculty from other educational institutions may use these materials for nonprofit educational purposes, provided this copyright notice is preserved.

Chapter 2 in EaC2e
Non-deterministic Finite Automata (NFAs)

Recognizer for an ILOC register name (allow redundant zeros)

Two Models for NFA Operation:

1. At any nondeterministic choice, the NFA clones itself and pursues all choices. If any clone terminates in a final state, the NFA accepts.

2. At any nondeterministic choice, the NFA makes the correct choice — that it, the one that leads to an accept.

Either model leads to the right intuitions.

And, to break the complexity models, epsilon transitions consume no input.

It is okay. We won’t execute NFAs

Note that this same model of execution also fits a DFA. It is overkill, but it works.
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
DFA versus NFA

Here is a DFA for \((a \mid b)^* \text{abb}\)

![DFA Diagram]

This DFA is not particularly obvious from the RE.

Each RE corresponds to one or more deterministic finite automatons (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.
DFA versus NFA

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

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 \(s_0\)
- In some way, the NFA model is more obviously expressive
Relationship between NFAs and DFAs

**DFA is a special case of an NFA**
- DFA has no $\varepsilon$ transitions
- DFA’s transition function is single-valued
- Rules for an NFA will correctly operate a DFA

**DFA can be simulated with an NFA**
  - *Obviously*

**NFA can be simulated with a DFA** *(less obvious, but still true)*
- Simulate sets of possible states
- Possible exponential blowup in the state space
- Still, one state per character in the input stream

$\Rightarrow$ **NFA & DFA** are equivalent in ability to recognize languages
The Plan for Scanner Construction

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

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

**DFA \(\rightarrow\) Minimal DFA**
- Hopcroft’s algorithm
- Brzozowski’s algorithm

**Minimal DFA \(\rightarrow\) Scanner**
- See § 2.5 in EaC2e

**DFA \(\rightarrow\) 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 **DFAs** 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 \( \varepsilon \) moves in precedence order & adjust final states

- **NFA for a**
- **NFA for \( ab \)**
- **NFA for \( a \mid b \)**
- **NFA for \( a^* \)**

Ken Thompson, CACM, 1968

**Precedence in REs:**
- Parentheses
- Closure
- Concatenation
- Alternation
Example of Thompson’s Construction

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

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

2. \( b \mid c \)

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

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

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

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

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

![NFA for $a$](image1)

![NFA for $ab$](image2)

![NFA for $a \mid b$](image3)

![NFA for $a^*$](image4)
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

---

The Cycle of Constructions
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 $\rightarrow$ DFA with Subset Construction

The subset construction builds a DFA that simulates the NFA

Two key support functions

- $\text{Move}(s_i, a)$ is the set of states reachable from $s_i$ by $a$
- $\text{FollowEpsilon}(s_i)$ 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...
NFA $\rightarrow$ DFA with Subset Construction

The algorithm:

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

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

\text{It tries each character in each } d_i. \text{ It builds every possible NFA configuration.}
\[\Rightarrow D \text{ and } T \text{ form the DFA}\]

This test is a little tricky

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

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

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

![Diagram of NFA and DFA with Subset Construction]

<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>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$c$</td>
<td></td>
</tr>
</tbody>
</table>

COMP 412, Fall 2019
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>$d_0$</td>
<td>$n_0$</td>
</tr>
</tbody>
</table>
NFA → DFA with Subset Construction

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

\[
\begin{array}{c}
\text{States} \\
\hline
\text{DFA} & \text{NFA} & a & b & c \\
\hline
d_0 & n_0 & n_1 & n_2 & n_3 & n_4 & n_6 & n_9 & \text{none} & \text{none} \\
\end{array}
\]
NFA → DFA with Subset Construction

a ( b | c )* :

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>a: $n_1$ $n_2$ $n_3$ $n_4$ $n_6$ $n_9$</td>
</tr>
<tr>
<td>$d_1$</td>
<td>$n_1$ $n_2$ $n_3$ $n_4$ $n_6$ $n_9$</td>
<td>b: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c: none</td>
</tr>
</tbody>
</table>
**NFA → DFA with Subset Construction**

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

![Diagram showing NFA and DFA transitions]

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

COMP 412, Fall 2019
a ( b | c )* : 

```
States
DFA NFA FollowEpsilon ( Move( s, * ) )
\hline
d_0 & n_0 & n_1 n_2 n_3 & n_4 n_6 n_9 & n_1 n_2 n_3 & n_4 n_6 n_9 & none & none 
d_1 & n_1 n_2 n_3 & n_4 n_6 n_9 & none & none & n_5 n_8 n_9 & n_3 n_4 n_6
\hline
```

COMP 412, Fall 2019
NFA → DFA with Subset Construction

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

\[ \text{States} \]

<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)</td>
<td>(n_4) (n_6) (n_9)</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)</td>
<td>(n_7) (n_8) (n_9)</td>
</tr>
</tbody>
</table>

COMP 412, Fall 2019
NFA $\rightarrow$ DFA with Subset Construction

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

![Diagram of NFA and DFA transition]

### 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>$n_1 \ n_2 \ n_3 \ n_4 \ n_6 \ n_9$</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$n_5 \ n_8 \ n_9 \ n_3 \ n_4 \ n_6$</td>
<td>$n_5 \ n_8 \ n_9 \ n_3 \ n_4 \ n_6$</td>
</tr>
</tbody>
</table>

COMP 412, Fall 2019
**NFA → DFA with Subset Construction**

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

\[
\begin{array}{c}
\text{States} \\
\text{DFA} & \text{NFA} \\
\hline
d_0 & n_0 \\
d_1 & n_1 \ n_2 \ n_3 \\
 & n_4 \ n_6 \ n_9 \\
d_2 & n_5 \ n_8 \ n_9 \\
 & n_3 \ n_4 \ n_6 \\
d_3 & n_7 \ n_8 \ n_9 \\
 & n_3 \ n_4 \ n_6 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{FollowEpsilon ( Move( s,* ) )} & a & b & c \\
\hline
\text{DFA} & \text{NFA} & n_1 \ n_2 \ n_3 & \text{none} & \text{none} \\
\hline
d_0 & n_0 & n_4 \ n_6 \ n_9 & \text{none} \\
d_1 & n_1 \ n_2 \ n_3 & \text{none} & n_5 \ n_8 \ n_9 \\
 & n_4 \ n_6 \ n_9 & n_3 \ n_4 \ n_6 & n_7 \ n_8 \ n_9 \\
d_2 & n_5 \ n_8 \ n_9 & n_3 \ n_4 \ n_6 & n_3 \ n_4 \ n_6 \\
d_3 & n_7 \ n_8 \ n_9 & n_3 \ n_4 \ n_6 & n_3 \ n_4 \ n_6 \\
\end{array}
\]
NFA → DFA with Subset Construction

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

![Diagram of NFA and DFA states and transitions]

### 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) none none</td>
</tr>
<tr>
<td>(d_1)</td>
<td>(n_1) (n_2) (n_3) (n_4) (n_6) (n_9) none</td>
<td>(n_5) (n_8) (n_9) (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) none</td>
<td>none</td>
</tr>
<tr>
<td>(d_3)</td>
<td>(n_7) (n_8) (n_9) (n_3) (n_4) (n_6) none</td>
<td>none</td>
</tr>
</tbody>
</table>
**NFA → DFA with Subset Construction**

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

![Diagram of NFA and DFA transition states]

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

\(n_7\) is the core state of \(d_3\)
**NFA \to DFA with Subset Construction**

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

\[
\begin{array}{c c c c c c c c c c c c c c c c c c c c}
\text{States} & \text{FollowEpsilon ( Move( s,\ast ))} \\
\hline
\text{DFA} & \text{NFA} & a & b & c \\
\hline
d_0 & n_0 & n_1 \ n_2 \ n_3 \ n_4 \ n_6 \ n_9 & \text{none} & \text{none} \\
d_1 & n_1 \ n_2 \ n_3 \ n_4 \ n_6 \ n_9 & \text{none} & n_5 \ n_8 \ n_9 & n_7 \ n_8 \ n_9 \\
d_2 & n_5 \ n_8 \ n_9 \ n_3 \ n_4 \ n_6 & \text{none} & d_2 & d_3 \\
d_3 & n_7 \ n_8 \ n_9 \ n_3 \ n_4 \ n_6 & \text{none} & d_2 & d_3 \\
\end{array}
\]

\[ n_5 \text{ is the core state of } d_2 \]
### NFA → DFA with Subset Construction

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

![Diagram of NFA and DFA](image)

### States

<table>
<thead>
<tr>
<th>DFA</th>
<th>NFA</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>d₀</td>
<td>n₀</td>
<td>n₁ n₂ n₃ n₄ n₆ n₉</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>d₁</td>
<td>n₁ n₂ n₃ n₄ n₆ n₉</td>
<td>none</td>
<td>n₅ n₈ n₉</td>
<td>n₇ n₈ n₉</td>
</tr>
<tr>
<td>d₂</td>
<td>n₅ n₈ n₉ n₃ n₄ n₆</td>
<td>none</td>
<td>d₂</td>
<td>d₃</td>
</tr>
<tr>
<td>d₃</td>
<td>n₇ n₈ n₉ n₃ n₄ n₆</td>
<td>none</td>
<td>d₂</td>
<td>d₃</td>
</tr>
</tbody>
</table>

The final states are because of state n₉.
NFA → DFA with Subset Construction

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

- **States**
  - DFA: \( d_0, d_1, d_2, d_3 \)
  - NFA: \( n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9 \)

- **Transition Table**

<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>( d_1 )</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>( d_2 )</td>
<td>( d_3 )</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>( d_3 )</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>( d_2 )</td>
<td>( d_3 )</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th></th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_0)</td>
<td>(d_1)</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>(d_1)</td>
<td>none</td>
<td>(d_2)</td>
<td>(d_3)</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>
machines 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 \( \mathfrak{A} = (S, M, S_0, F) \) be a nondeterministic automaton. \( \mathfrak{T}(\mathfrak{A}) \) 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 \( N(t, \sigma) \) is the union of the sets \( M(s, \sigma) \) for \( s \) in \( t \), \( t_0 = S_0 \), and \( G \) is the set of all subsets of \( S \) containing at least one member of \( F \).

Clearly \( \mathfrak{T}(\mathfrak{A}) \) is an ordinary automaton, but it is actually equivalent to \( \mathfrak{A} \).

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

**Proof:** Assume first that \( T(\mathfrak{A}) \) and \( T(\mathfrak{T}(\mathfrak{A})) \) contain states satisfying the conditions of Definition 10. We show by induction that for \( k \leq n \), \( s_k \) is in \( N(t_0, x_k) \). For \( k = 0 \), \( N(t_0, x_0) = N(t_0, \Lambda) = t_0 = S_0 \) and we were given that \( s_0 \) is in \( S_0 \). Assume the result for \( k - 1 \). By definition, \( N(t_0, x_k) = N(N(t_0, x_{k-1}), \sigma_{k-1}) \). But we have assumed \( s_{k-1} \) is in \( N(t_0, x_{k-1}) \) so that from the definition of \( N \) we have \( M(s_{k-1}, \sigma_{k-1}) \subseteq N(t_0, x_{k-1}) \). However, \( s_k \) is in \( M(s_{k-1}, \sigma_{k-1}) \) and so the result is established. In particular, \( s_n \) is in \( N(t_0, x) = N(t_0, x) \), and since \( s_n \) is in \( F \), we have \( N(t_0, x) \) in \( G \), which proves that \( x \) is in \( T(\mathfrak{T}(\mathfrak{A})) \). Hence, we have shown that \( T(\mathfrak{A}) \subseteq T(\mathfrak{T}(\mathfrak{A})) \).

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

**Definition 12.** Let \( \mathfrak{A} = (S, M, S_0, F) \) be a nondeterministic automaton. The dual of \( \mathfrak{A} \) is the machine \( \mathfrak{A}^* = (S, M^*, F, S_0) \) where the function \( M^* \) is defined by the condition

\[
\sigma' \text{ is in } M^*(s, \sigma) \text{ if and only if } s \text{ is in } M(s', \sigma).
\]

Notice that we have at once the equation \( \mathfrak{A}^{**} = \mathfrak{A} \).

The relation between the sets defined by an automaton and its dual is as follows.

**Theorem 12.** If \( \mathfrak{A} \) is a nondeterministic automaton, then \( T(\mathfrak{A}^*) = T(\mathfrak{A})^* \).

**Proof:** In view of the equality \( \mathfrak{A}^{**} = \mathfrak{A} \), we need only show \( T(\mathfrak{A}^*) \subseteq T(\mathfrak{A})^* \). Let \( x = \sigma_0 \sigma_1 \ldots \sigma_{n-1} \) be a tape in \( T(\mathfrak{A})^* \). Show that \( x^* \) is in \( T(\mathfrak{A})^* \). Let \( s_0, s_1, \ldots, s_n \) be any internal states of \( \mathfrak{A}^* \) such that \( s_0 \) and \( s_k \) is in \( M^*(s_{k-1}, \sigma_{k-1}) \) for \( k = 1, 2, \ldots, n \). Define a new sequence \( s'_0, s'_1, \ldots, s'_n \) by the equation \( s'_k = s_{k-n} \) for \( k \leq n \). Obviously, \( s'_0 \) is in \( S_0 \) and \( s'_n \) is in \( F \). Further, for \( k > 0 \) and \( k \leq n \), \( s'_{k-1} = s_{n-k} \) is in \( M^*(s_{n-k}, \sigma_{n-k}) \), or in other words, \( s'_{n-k} = s'_{n-k} \) is in \( M(s'_{k-1}, \sigma_{n-k}) \). Now defining a new sequence of symbols \( \sigma'_{0} \sigma'_{1} \ldots \sigma'_{n-1} \) by the formula \( \sigma'_{k} = \sigma_{n-k} \), we see that \( \sigma'_{n-k} = \sigma_{n-k} \) and \( \sigma'_{0} \sigma'_{1} \ldots \sigma'_{n-1} = x^* \). Thus, \( x^* \) is in \( T(\mathfrak{A})^* \) as was to be proved.

It should be noted that Theorem 12 together with Theorem 11 yields a direct construction and proof for Theorem 4 of Section 3 which was first proved by the indirect method of Theorem 1. In the next section we make heavy use of the direct constructions supplied by the nondeterministic machines to obtain results not easily apparent from the mathematical characterizations of Theorems 1 and 2.

### 6. Further closure properties
Simplifying a result due originally to Kleene, Myhill in unpublished work has shown that the class \( \mathcal{C} \) can be characterized as the least class of sets of tapes containing the finite sets and closed under some simple operations on sets of tapes. We indicate here a different proof using...
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 $\varepsilon$-transitions between each single-character automaton

Subset construction eliminates $\varepsilon$-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,

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

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 \( \text{reverse}(NFA) \) merges suffixes in original NFA
Brzozowski’s Algorithm

Step 2

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

The Cycle of Constructions

Minimal DFA
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
Thompson’s construction produces something along these lines

To make the example fit, we have eliminated some of the $\varepsilon$-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
Abbreviated Register Specification

Brzozowski, step 2 reverses that DFA and subsets it again

A skilled human might build this DFA

0,1,2,3,4, 5,6,7,8,9

The Cycle of Constructions

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

```
for i ← 0 to |D| - 1;  // label each immediate path
    for j ← 0 to |D| - 1;
        R^0_{ij} ← \{ a | \delta(d_i, a) = d_j \};
        if (i = j) then
            R^0_{ii} = R^0_{ii} \cup \{ \epsilon \};
for k ← 0 to |D| - 1;  // label nontrivial paths
    for i ← 0 to |D| - 1;
        for j ← 0 to |D| - 1;
            R^k_{ij} ← R^{k-1}_{ik} (R^{k-1}_{kk})* R^{k-1}_{kj} / R^{k-1}_{ij}
L ← \{ \}  // union labels of paths from
For each final state s_i  // s_0 to a final state s_i
    L ← L \cup R^{|D| - 1}_{0i}
```

- **R^k_{ij}** is the set of paths from **i** to **j** that include no state higher than **k**

### The Cycle of Constructions

Adaptation of all points, all paths, low cost algorithm

COMP 412, Fall 2019