In linear time from Regular Expressions to NFAs

Thompson’s Algorithm
Produces \( O(n) \) states for regular expressions of length \( n \).

Berkeley-Sethi Approach

Berry-Sethi Approach

Berry-Sethi Algorithm
Produces exactly \( n + 1 \) states without \( \epsilon \)-transitions and demonstrates \( \rightarrow \text{Equality Systems} \) and \( \rightarrow \text{Attribute Grammars} \)

Glushkov Algorithm
Produces exactly \( n + 1 \) states without \( \epsilon \)-transitions and demonstrates \( \rightarrow \text{Equality Systems} \) and \( \rightarrow \text{Attribute Grammars} \)

Idea:
The automaton tracks (conventionally via a marker "*"), in the syntax tree of a regular expression, which subexpressions in \( \epsilon \) are reachable consuming the rest of input \( w \).
Berry-Sethi Approach

... for example:

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

![Diagram](image)

Berry-Sethi Approach

... for example:

\[w = \emptyset baa :\]

![Diagram](image)
Berry-Sethi Approach

... for example:

\[ w = \ldots a \]

\[ \begin{array}{c}
\ast \\
\ast \\
\ast
\end{array}
\]

\[ \begin{array}{c}
a \\
b \\
a \\
b
\end{array}
\]

Berry-Sethi Approach

... for example:

\[ w = a \]

\[ \begin{array}{c}
\ast \\
\ast \\
\ast
\end{array}
\]

\[ \begin{array}{c}
a \\
b \\
a \\
b
\end{array}
\]

Berry-Sethi Approach

... for example:

\[ w = \ldots \]

\[ \begin{array}{c}
\ast \\
\ast \\
\ast
\end{array}
\]

\[ \begin{array}{c}
a \\
b \\
a \\
b
\end{array}
\]

Berry-Sethi Approach

... for example:

\[ w = \ldots \]

\[ \begin{array}{c}
\ast \\
\ast \\
\ast
\end{array}
\]

\[ \begin{array}{c}
a \\
b \\
a \\
b
\end{array}
\]
Berry-Sethi Approach

In general:

- Input is only consumed at the leaves.
- Navigating the tree does not consume input \( \rightarrow \epsilon \)-transitions
- For a formal construction we need identifiers for states.
- For a node \( n \)'s identifier we take the subexpression, corresponding to the subtree dominated by \( n \).
- There are possibly identical subexpressions in one regular expression.

\[ \text{we enumerate the leaves ...} \]

... for example:

```
\[
\begin{array}{c}
\text{a} \\
\text{b}
\end{array}
\]
```

```
\[
\begin{array}{c}
\text{a} \\
\text{b}
\end{array}
\]
```

```
\[
\begin{array}{c}
\text{a} \\
\text{b}
\end{array}
\]
```

```
\[
\begin{array}{c}
\text{a} \\
\text{b}
\end{array}
\]
```

```
\[
\begin{array}{c}
\text{a} \\
\text{b}
\end{array}
\]
```

```
\[
\begin{array}{c}
\text{a} \\
\text{b}
\end{array}
\]
```

```
\[
\begin{array}{c}
\text{a} \\
\text{b}
\end{array}
\]
```
Berry-Sethi Approach (naive version)

Construction (naive version):

States: $\bullet r, \bullet e$ with $r$ nodes of $e$;  
Start state: $\bullet e$;  
Final state: $\bullet e$;  
Transitions: for leaves $r \equiv l \mid x$ we require: $(\bullet r, x, \bullet e)$.  
The leftover transitions are:

<table>
<thead>
<tr>
<th>$r$</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \mid r_2$</td>
<td>$(\bullet r, e, \bullet r_1)$, $(\bullet r, e, \bullet r_2)$, $(\bullet r_1, e, \bullet r_1)$, $(\bullet r_2, e, \bullet r)$</td>
</tr>
</tbody>
</table>

Berry-Sethi Approach

Discussion:

- Most transitions navigate through the expression  
- The resulting automaton is in general nondeterministic

⇒ Strategy for the sophisticated version: Avoid generating $\epsilon$-transitions

Berry-Sethi Approach (naive version)

Construction (naive version):

States: $\bullet r, \bullet e$ with $r$ nodes of $e$;  
Start state: $\bullet e$;  
Final state: $\bullet e$;  
Transitions: for leaves $r \equiv l \mid x$ we require: $(\bullet r, x, \bullet e)$.  
The leftover transitions are:

<table>
<thead>
<tr>
<th>$r$</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 \mid r_2$</td>
<td>$(\bullet r, e, \bullet r_1)$, $(\bullet r, e, \bullet r_2)$, $(\bullet r_1, e, \bullet r_1)$, $(\bullet r_2, e, \bullet r)$</td>
</tr>
</tbody>
</table>

| $r_1 \mid r_2$ | $(\bullet r, e, \bullet r_1)$, $(\bullet r, e, \bullet r_2)$, $(\bullet r_1, e, \bullet r_1)$, $(\bullet r_2, e, \bullet r)$ |
Berry-Sethi Approach

Discussion:
- Most transitions navigate through the expression
- The resulting automaton is in general nondeterministic

⇒ Strategy for the sophisticated version:
Avoid generating $\epsilon$-transitions

Idea:
Pre-compute helper attributes during $D(\text{depth})F(\text{first})S(\text{search})$!

Berry-Sethi Approach: 1st step

$\text{empty}[r] = t$ if and only if $\epsilon \in [r]$

... for example:

```
  * 0
d 2
d 1
```

Berry-Sethi Approach: 1st step

$\text{empty}[r] = t$ if and only if $\epsilon \in [r]$

... for example:

```
  * 0
d 2
d 1
```

Necessary node-attributes:

- **first** the set of read states below $r$, which may be reached first, when descending into $r$.
- **next** the set of read states to the right of $r$, which may be reached first in the traversal after $r$.
- **last** the set of read states below $r$, which may be reached last when descending into $r$.
- **empty** can the subexpression $r$ consume $\epsilon$?
Berry-Sethi Approach: 1st step

\[ \text{empty}[r] = \top \text{ if and only if } \epsilon \in [r] \]

... for example:

```
      f
     / \
    *   *
   /   /   \
  f   f   f
  /   /   /   \   \  \
 f   f   f   2   3   4
  /   /   /   /   /   /   \
 f   f   f   0   1   3   4
```

Berry-Sethi Approach: 2nd step

The may-set of first reached read states: The set of read states, that may be reached from \( \bullet r \) (i.e. while descending into \( r \)) via sequences of \( \epsilon \)-transitions:

\[ \text{first}[r] = \{ i \in r \mid (\bullet r, \epsilon, i, x) \in \delta^*, x \neq \epsilon \} \]

... for example:

```
      f
     / \
    *   *
   /   /   \
  f   f   f
  /   /   /   \   \  \
 f   f   f   2   3   4
  /   /   /   /   /   /   \
 f   f   f   0   1   3   4
```

Implementation:

DFS post-order traversal

for leaves \( r = [x] \) we find \( \text{empty}[r] = (x = \epsilon) \).

Otherwise:

\[
\begin{align*}
\text{empty}[r \cup r_2] &= \text{empty}[r] \lor \text{empty}[r_2] \\
\text{empty}[r \cdot r_2] &= \text{empty}[r] \land \text{empty}[r_2] \\
\text{empty}[r_1] &= \top \\
\text{empty}[r_1?] &= \top
\end{align*}
\]

Berry-Sethi Approach: 2nd step

The may-set of first reached read states: The set of read states, that may be reached from \( \bullet r \) (i.e. while descending into \( r \)) via sequences of \( \epsilon \)-transitions:

\[ \text{first}[r] = \{ i \in r \mid (\bullet r, \epsilon, i, x) \in \delta^*, x \neq \epsilon \} \]

... for example:

```
      f
     / \
    *   *
   /   /   \
  f   f   f
  /   /   /   \   \  \
 f   f   f   2   3   4
  /   /   /   /   /   /   \
 f   f   f   0   1   3   4
```
Berry-Sethi Approach: 2nd step

The may-set of first reached read states: The set of read states, that may be reached from \( \star \) (i.e. while descending into \( r \)) via sequences of \( \epsilon \)-transitions:

\[
\text{first}[r] = \{ i \in r \mid (\star, \epsilon, i, x) \in \delta, x \neq \epsilon \}
\]

... for example:

---

Berry-Sethi Approach: 2nd step

The may-set of first reached read states: The set of read states, that may be reached from \( \star \) (i.e. while descending into \( r \)) via sequences of \( \epsilon \)-transitions:

\[
\text{first}[r] = \{ i \in r \mid (\star, \epsilon, i, x) \in \delta, x \neq \epsilon \}
\]

... for example:

---

Implementation:

DFS post-order traversal

For leaves \( r = \{ x \} \) we find \( \text{first}[r] = \{ i \mid x \neq \epsilon \} \).

Otherwise:

\[
\begin{align*}
\text{first}[r_1 \cdot r_2] &= \text{first}[r_1] \cup \text{first}[r_2] \\
\text{first}[r_1 | r_2] &= \begin{cases} 
\text{first}[r_1] \cup \text{first}[r_2] & \text{if empty}(r_1) = i \\
\text{first}[r_1] & \text{if empty}(r_1) = \not{\exists} i
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\text{first}[r_1 \cdot | r_2] &= \text{first}[r_1] \\
\text{first}[r_1 | r_2] &= \text{first}[r_1]
\end{align*}
\]
Berry-Sethi Approach: 3rd step

The may-set of next read states: The set of read states within the subtrees right of $r^*$, that may be reached next via sequences of $\epsilon$-transitions.

$\text{next}[r] = \{ i \mid \langle r^* \epsilon, \epsilon, \epsilon \rangle \in \delta^*, x \neq \epsilon \}$

... for example:

```
      012
     /   \
01 ---+--- 2
 |     |   |
01 f  2 f 3 f
0 f   2 f 3 f
0 a   2 a 3 a
```

Berry-Sethi Approach: 3rd step

The may-set of next read states: The set of read states within the subtrees right of $r^*$, that may be reached next via sequences of $\epsilon$-transitions.

$\text{next}[r] = \{ i \mid \langle r^* \epsilon, \epsilon, \epsilon \rangle \in \delta^*, x \neq \epsilon \}$

... for example:

```
      012
     /   \
01 ---+--- 2
 |     |   |
01 f  2 f 3 f
0 f   2 f 3 f
0 a   2 a 3 a
```

Berry-Sethi Approach: 3rd step

The may-set of next read states: The set of read states within the subtrees right of $r^*$, that may be reached next via sequences of $\epsilon$-transitions.

$\text{next}[r] = \{ i \mid \langle r^* \epsilon, \epsilon, \epsilon \rangle \in \delta^*, x \neq \epsilon \}$

... for example:

```
      012
     /   \
01 ---+--- 2
 |     |   |
01 f  2 f 3 f
0 f   2 f 3 f
0 a   2 a 3 a
```

Berry-Sethi Approach: 3rd step

The may-set of next read states: The set of read states within the subtrees right of $r^*$, that may be reached next via sequences of $\epsilon$-transitions.

$\text{next}[r] = \{ i \mid \langle r^* \epsilon, \epsilon, \epsilon \rangle \in \delta^*, x \neq \epsilon \}$

... for example:

```
      012
     /   \
01 ---+--- 2
 |     |   |
01 f  2 f 3 f
0 f   2 f 3 f
0 a   2 a 3 a
```
Berry-Sethi Approach: 3rd step

The *may-set* of next read states: The set of read states within the subtrees right of \( r \), that may be reached next via sequences of \( \epsilon \)-transitions.  
\[
\text{next}\{r\} = \{ i \mid (r, \epsilon, i, x) \in \delta^*, x \neq \epsilon \}
\]

... for example:

```
      *  
     /   
    *    
   /     
  01    01 
   /     / 
  f     f  
```

Berry-Sethi Approach: 4th step

The *may-set* of last reached read states: The set of read states, which may be reached last during the traversal of \( r \) connected to the root via \( \epsilon \)-transitions only:  
\[
\text{last}\{r\} = \{ i \in r \mid (i, x, \epsilon, r) \in \delta^*, x \neq \epsilon \}
\]

... for example:

```
      *  
     /   
    *    
   /     
  01    01 
   /     / 
  f     f  
```

Implementation:
DFS pre-order traversal

For the root, we find:  
\[
\text{next}\{r\} = \emptyset
\]

Apart from that we distinguish, based on the context:

<table>
<thead>
<tr>
<th>( r )</th>
<th>Equalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 )</td>
<td>( \text{next}{r_1} = \text{next}{r} )</td>
</tr>
<tr>
<td>( r_2 )</td>
<td>( \text{next}{r_2} = \text{next}{r} )</td>
</tr>
</tbody>
</table>
| \( r_1 \cdot r_2 \) | \( \text{next}\{r_1 \cdot r_2\} = \text{first}\{r_1\} \cup \text{next}\{r\} \) if \( \text{empty}\{r_2\} = t \)  
\[
\text{next}\{r_2\} = \text{next}\{r\}
\]
| \( r_1 \cdot r_2 \) | \( \text{next}\{r_1 \cdot r_2\} = \text{first}\{r_1\} \cup \text{next}\{r\} \) if \( \text{empty}\{r_2\} = f \) |
| \( r_1 \cdot r_2 \) | \( \text{next}\{r_1 \cdot r_2\} = \text{next}\{r\} \) |
| \( r_1 \) | \( \text{next}\{r_1\} = \text{first}\{r\} \cup \text{next}\{r\} \) |

Berry-Sethi Approach: 4th step

The *may-set* of last reached read states: The set of read states, which may be reached last during the traversal of \( r \) connected to the root via \( \epsilon \)-transitions only:  
\[
\text{last}\{r\} = \{ i \in r \mid (i, x, \epsilon, r) \in \delta^*, x \neq \epsilon \}
\]

... for example:

```
      *  
     /   
    *    
   /     
  01    01 
   /     / 
  f     f  
```

```
      *  
     /   
    *    
   /     
  01    01 
   /     / 
  f     f  
```
**Berry-Sethi Approach: 4th step**

**Implementation:**

DFS *post-order* traversal

for leaves \( r = \{ i | x \neq e \} \) we find \( \text{last}[r] = \{ i | x \neq e \} \).

Otherwise:

\[
\begin{align*}
\text{last}[r_1 \cdot r_2] &= \text{last}[r_1] \cup \text{last}[r_2] \\
\text{last}[r] &= \begin{cases} 
\text{last}[r_1] \cup \text{last}[r_2] & \text{if empty}[r_2] = i \\
\text{last}[r_2] & \text{if empty}[r_1] = i
\end{cases}
\end{align*}
\]

**Berry-Sethi Approach**

... for example:

![Diagram](image)

**Remarks:**

- This construction is known as Berry-Sethi- or Glushkov-construction.
- It is used for XML to define Content Models
- The result may not be, what we had in mind...

**Berry-Sethi Approach: (sophisticated version)**

Construction (sophisticated version): Create an automaton based on the syntax tree’s new attributes:

**States:** \( \{ \ast \} \cup \{ i | i \text{ a leaf} \} \)

**Start state:** \( \ast \)

**Final states:** \( \text{last}[e] \) \( i \) \( \text{if empty}[e] = i \)

\( \{ \ast \} \cup \text{last}[e] \) \( \text{if empty}[e] = \ast \)

**Transitions:**

- \( \ast - a - i \) if \( i \in \text{first}[e] \) and \( i \) labeled with \( a \).
- \( \ast - a - i \) if \( i \in \text{next}[e] \) and \( i \) labeled with \( a \).

We call the resulting automaton \( A_c \).

**The expected outcome:**

![Diagram](image)

**Remarks:**

- ideal automaton would be even more compact
- but Berry-Sethi is rather directly constructed
- Anyway, we need a deterministic version

\( \Rightarrow \) Powerset-Construction
Berry-Sethi Approach

... for example:

Remarks:
- This construction is known as Berry-Sethi- or Glushkov-construction.
- It is used for XML to define Content Models
- The result may not be, what we had in mind...

The expected outcome:

Remarks:
- ideal automaton would be even more compact
- but Berry-Sethi is rather directly constructed
- Anyway, we need a deterministic version

⇒ Powerset-Construction

Lexical Analysis

Chapter 4:

Turning NFAs deterministic

Powerset Construction

... for example:
Powerset Construction

... for example:

Powerset Construction

... for example:

Theorem:

For every non-deterministic automaton \( A = (Q, \Sigma, \delta, I, F) \) we can compute a deterministic automaton \( \mathcal{P}(A) \) with

\[
\mathcal{L}(A) = \mathcal{L}(\mathcal{P}(A))
\]
Powerset Construction

**Theorem:**
For every non-deterministic automaton \( A = (Q, \Sigma, \delta, I, F) \) we can compute a deterministic automaton \( \mathcal{P}(A) \) with
\[
\mathcal{L}(A) = \mathcal{L}(\mathcal{P}(A))
\]

**Construction:**

- **States:** Powersets of \( Q \);
- **Start state:** \( I \);
- **Final states:** \( \{Q' \subseteq Q \mid Q' \cap F \neq \emptyset\} \);
- **Transitions:**
  \[
  \delta_P(Q', a) = \{ q \in Q \mid \exists p \in Q': [p, a, q] \in \delta \}.
  \]

Powerset Construction

**Bummer!**
There are exponentially many powersets of \( Q \)

- Idea: Consider only contributing powersets. Starting with the set \( Q_P = \{I\} \) we only add further states by need ... 
- i.e., whenever we can reach them from a state in \( Q_P \)
- Even though, the resulting automaton can become enormously huge 
  ... which is (sort of) not happening in practice

- Therefore, in tools like grep a regular expression's DFA is never created!
- Instead, only the sets, directly necessary for interpreting the input are generated while processing the input

... for example:

```plaintext
0 b 1 a
a 2 a
b 3 a
b a
```

Powerset Construction

**Bummer!**
There are exponentially many powersets of \( Q \)

- Idea: Consider only contributing powersets. Starting with the set \( Q_P = \{I\} \) we only add further states by need ... 
- i.e., whenever we can reach them from a state in \( Q_P \)
- Even though, the resulting automaton can become enormously huge 
  ... which is (sort of) not happening in practice

- Therefore, in tools like grep a regular expression's DFA is never created!
- Instead, only the sets, directly necessary for interpreting the input are generated while processing the input

```plaintext
0 2 3
b a
```
Powerset Construction

... for example:

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{a} \\
\text{b}
\end{array}
\]

\[
\begin{array}{c}
\text{0} \\
\text{2} \\
\text{1} \\
\text{4}
\end{array}
\]

\[
\begin{array}{c}
\text{a} \\
\text{a} \\
\text{a} \\
\text{a}
\end{array}
\]

\[
\begin{array}{c}
\text{b} \\
\text{b} \\
\text{a} \\
\text{b}
\end{array}
\]

\[
\begin{array}{c}
\text{02} \\
\text{023} \\
\text{1} \\
\text{14}
\end{array}
\]

Powerset Construction

... for example:

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{a} \\
\text{b}
\end{array}
\]

\[
\begin{array}{c}
\text{0} \\
\text{2} \\
\text{1} \\
\text{4}
\end{array}
\]

\[
\begin{array}{c}
\text{a} \\
\text{a} \\
\text{a} \\
\text{a}
\end{array}
\]

\[
\begin{array}{c}
\text{b} \\
\text{b} \\
\text{a} \\
\text{b}
\end{array}
\]

\[
\begin{array}{c}
\text{02} \\
\text{023} \\
\text{1} \\
\text{14}
\end{array}
\]

Remarks:

- For an input sequence of length \( n \), maximally \( O(n) \) sets are generated.
- Once a set/edge of the DFA is generated, they are stored within a hash-table.
- Before generating a new transition, we check this table for already existing edges with the desired label.
**Powerset Construction**

... for example:

\[ a \ b \ a \ b \]

**Remarks:**

- For an input sequence of length \( n \), maximally \( O(n) \) sets are generated.
- Once a set/edge of the DFA is generated, they are stored within a hash-table.
- Before generating a new transition, we check this table for already existing edges with the desired label.

**Lexical Analysis**

**Chapter 5:**

**Scanner design**