"If enlightenment doesn't come immediately, you need practice. It's like Buddhism."

Thorsten Altenkirch
teaching Machines and their Languages
at University of Nottingham

Nondeterministic vs. Deterministic Automata

Formal Definition of NFA
NFA is a mathematical model defined as a quintuple:

\[ M = (Q, \Sigma, \delta, q_0, F) \]

- \( Q \): Set of states, i.e. \( \{q_0, q_1, q_2\} \)
- \( \Sigma \): Input alphabet, i.e. \( \{a, b\} \)
- \( \delta \): Transition function
- \( q_0 \): Initial state
- \( F \): Final (accepting) states

Deterministic Finite Automata

A deterministic finite automata (DFA) is a special case of a nondeterministic finite automaton in which

1. no state has a \( \lambda \)-transition, i.e., a transition on empty input string \( \lambda \), and
2. for each state \( q \) and input symbol \( a \), there is at most one edge labeled \( a \) leaving \( q \).

Example

A nondeterministic finite automaton

Transition table for the finite automaton above

Example

NFA accepting \( aa^* + bb^* \)

Example

NFA accepting \((a+b)^*abb\)
NFA recognizing three different patterns.

(a) NFA for $a$, $abb$, and $a^*b$.

(b) Combined NFA.

Example

NFA recognizing keyword IF

Ways to think of nondeterminism

- always make the correct guess when several choices (Oracle, not a practical solution)
- “backtracking” (systematically try all possibilities)

For a particular string, imagine a tree of all possible state transitions:

Advantages of nondeterminism

- a NFA can be smaller, easier to design and easier to understand than a DFA for the same language
- NFA useful for theorem proving (compact notation)
- NFA are a good introduction to nondeterminism in more powerful computational models, where nondeterminism plays an important role (e.g. NPDA, non-deterministic push down automata)

Determinism vs. nondeterminism

<table>
<thead>
<tr>
<th>AUTOMATON</th>
<th>SPACE</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFA</td>
<td>$O(</td>
<td>r</td>
</tr>
<tr>
<td>DFA</td>
<td>$O(2^{</td>
<td>r</td>
</tr>
</tbody>
</table>

Space and time taken to recognize regular expressions:
NFA more compact but take time to backtrack all choices
DFA take place, but save time

Equivalent automata

Two finite automata $M_1$ and $M_2$ are equivalent if

$$L(M_1) = L(M_2),$$

that is, if they both accept the same language.

Equivalence of NFAs and DFAs

To show that NFAs and DFAs accept the same class of languages, we show two things:

- any language accepted by a DFA can also be accepted by some NFA (As DFA is a special case of NFA)
- any language accepted by a NFA can also be accepted by some (corresponding, specially constructed) DFA

Proof strategy

To show that any language accepted by a NFA is also accepted by some DFA, we describe an algorithm that takes any NFA and converts it into a DFA that accepts the same language

The algorithm is called the “subset construction algorithm”

We can use mathematical induction (on the length of a string accepted by the automaton) to prove that the DFA that is constructed accepts the same language as the NFA.

Converting NFA to DFA

Subset Construction
Subset construction

What does it do?
Given a NFA, it constructs a DFA that accepts the same language.

What is the key idea?
The equivalent DFA simulates the NFA by keeping track of the possible states it could be in. Each state of the DFA is a subset of the set of states of the NFA - hence, the name of the algorithm.

If the NFA has N states, the DFA can have as many as $2^N$ states, although it usually has many less.

Steps of subset construction

The initial state of the DFA is the set of all states the NFA can be in without reading any input.

For any state $\{q_1, q_2, \ldots, q_k\}$ of the DFA and any input $a$, the next state of the DFA is the set of all states of the NFA that can result as next states if the NFA is in any of the states $q_1, q_2, \ldots, q_k$ when it reads $a$. This includes states that can be reached by reading $a$, followed by any number of $\lambda$-moves. Use this rule to keep adding new states and transitions until it is no longer possible to do so.

The accepting states of the DFA are those states that contain at least one accepting state of the NFA.

Example

Here is a NFA that we want to convert to an equivalent DFA.

---

Subset Construction

Algorithm

For state $\{2\}$, we create a transition for each possible input, a and b. From $\{2\}$, with b we are either back to $\{2\}$ (loop) or we reach $\{1\}$ - see the little framed original NFA. So from $\{2\}$, with b we end in state $\{1, 2\}$. Reading a leads us from $\{2\}$ to $\{0\}$ in the original NFA, which means state $\{0, 1\}$ in the new DFA.

For state $\{1, 2\}$, we make again transition for each possible input, a and b. From $\{1, 2\}$ a leads us to $\{0\}$. From $\{1\}$ with a we are back to $\{1\}. So, we reach $\{0, 1\}$ with a from $\{1, 2\}$. With b we are back to $\{1, 2\}$.

At this point, a transition is defined for every state-input pair.

The last step is to mark the final states of the DFA. As $\{1\}$ was the accepting state in NFA, all states containing $\{1\}$ in DFA will be accepting states: $\{0, 1\}$ and $\{1, 2\}$.
Subset Construction

States of nondeterministic \( M' \) will correspond to set of states of deterministic \( M \)

Where \( q_0 \) is start state of \( M' \), use \( \{q_0\} \) as start state of \( M \).

Accepting states of \( M' \) will be those state-sets containing at least one accepting state of \( M \).

Accepting states of \( M' \) will be those state-sets containing at least one accepting state of \( M \).

Eliminate any state-set, as well as all edges incident upon it, such that there is no path leading to it from \( \{q_0\} \).

The power set of a finite set, \( Q \), consists of \( 2^{|Q|} \) elements

...which means that the DFA corresponding to a given NFA with \( Q \) states have a finite number of states, \( 2^{|Q|} \).

If \( |Q| = 0 \) then \( Q \) is the empty set, \( |P(Q)| = 1 = 2^0 \).

If \( |Q| = N \) and \( N \geq 1 \), the size of the power set is always twice as large as it would be for a set with one less element.

\[ |P(A)| = 2 \cdot |P(B)|, \] where \( |B| = |A| - 1 \).

We construct subset of a given set so that for each element of the initial set there are two alternatives, either is the element member of a subset or not. So we have

\[
2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot \ldots \cdot 2 = 2^N
\]

N times

From an NFA to a DFA

<table>
<thead>
<tr>
<th>Subset Construction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )-closure(s)</td>
<td>Set of NFA states reachable from an NFA state ( s ) on ( \lambda )-transitions along</td>
</tr>
<tr>
<td>( \lambda )-closure(T)</td>
<td>Set of NFA states reachable from some NFA state ( s ) in ( T ) on ( \lambda )-transitions along</td>
</tr>
<tr>
<td>Move(T,a)</td>
<td>Set of NFA states reachable from some NFA state ( s ) set with a transition on input symbol ( a )</td>
</tr>
</tbody>
</table>

Finite Language Theorem

Any finite language is FSA-acceptable (regular).

Application

Lexical Analysis

Phases of a compiler

Scanner

Lexical Analyzer

Tokens

Parser

Syntax Analyzer

Parse Tree

Semantic Analyzer

Abstract Syntax Tree with attributes
The Cycle of Constructions

RE → NFA (Thompson’s construction)
Build an NFA for each term
Combine them with ε-moves
NFA → DFA (Subset construction)
DFA → Minimal DFA
DFA → RE
All pairs, all paths problem
Union together paths from s₀ to a final state

Some Properties of Regular Languages

For regular languages \( L_1 \) and \( L_2 \)
we will prove that:

- **Union:** \( L_1 \cup L_2 \)
- **Concatenation:** \( L_1 L_2 \)
- **Kleene star:** \( L_1^* \)

Regular languages are closed under

\[ (L_1 \cup L_2)^* \]

This means that from two original automata representing regular languages we will be able to construct the corresponding new automaton for union, concatenation and Kleene star operations.

Example

\[ L_1 = \{ a^n b \} \]
\[ L_2 = \{ ba \} \]

\[ (L_1 \cup L_2)^* \]
Example
NFA for \( L_1 \cup L_2 = \{a^n b\} \cup \{ba\} \)

\[
L_1 = \{a^n b\} \\
L_2 = \{ba\}
\]

Example
Concatenation
NFA for \( L_1 L_2 \)

\[
M_1 \\
M_2
\]

Example
NFA for \( L_4 L_2 = \{a^n b\} \{ba\} = \{a^n bba\} \)

\[
L_4 = \{a^n b\} \\
L_2 = \{ba\}
\]

Summary: Operations on Regular Expressions

<table>
<thead>
<tr>
<th>RE</th>
<th>Regular language Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a+b</td>
<td>{a, b}</td>
</tr>
<tr>
<td>(a+b)(a+b)</td>
<td>{aa, ab, ba, bb}</td>
</tr>
<tr>
<td>a*</td>
<td>{λ, a, aa, aaaa, ...}</td>
</tr>
<tr>
<td>a*b</td>
<td>{b, ab, aab, aaab, ...}</td>
</tr>
<tr>
<td>(a+b)*</td>
<td>{λ, a, b, aa, ab, ba, aaaa, bbbb, ...}</td>
</tr>
</tbody>
</table>

Algebraic Properties

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r + s = s + r)</td>
<td>is commutative</td>
</tr>
<tr>
<td>(r + (s + t) = (r + s) + t)</td>
<td>is associative</td>
</tr>
<tr>
<td>((rs)t = r(st))</td>
<td>concatenation is associative</td>
</tr>
<tr>
<td>((r + s)t = rs + rt)</td>
<td>concatenation distributes over +</td>
</tr>
<tr>
<td>(λs = rs)</td>
<td>(λ) is the identity element for concatenation</td>
</tr>
<tr>
<td>(ra = ar)</td>
<td></td>
</tr>
<tr>
<td>(rλ = r)</td>
<td></td>
</tr>
<tr>
<td>(r^* = (r + λ)^*)</td>
<td>relation between (^*) and (λ)</td>
</tr>
<tr>
<td>(r^* = r^*)</td>
<td>(*) is idempotent</td>
</tr>
</tbody>
</table>

Example: From a Regular Expression to an NFA
Example: \((a+b)*abb\) step by step construction

Example: From a Regular Expression to an NFA
Example: \((a+b)*abb\)
Example: From a Regular Expression to an NFA

Example: \((a+b)^*abb\)

```
Example: From a Regular Expression to an NFA
Example: (a+b)*abb
```

### Theorem - Part 1

**Languages Generated by Regular Expressions** \(\subseteq\) **Regular Languages**

For any regular expression \(r\) the language \(L(r)\) is regular

### Theorem - Part 2

**Languages Generated by Regular Expressions** \(\supseteq\) **Regular Languages**

For any regular language \(L\) there is a regular expression \(r\) such that \(L(r) = L\)

### Proof - Part 1

For any regular expression \(r\) the language \(L(r)\) is regular

Proof by induction on the size of \(r\)

### Induction Basis

Primitive Regular Expressions: \(\emptyset, \lambda, \alpha\)

- \(L(M_1) = \emptyset = L(\emptyset)\)
- \(L(M_2) = \{\lambda\} = L(\lambda)\) Regular Languages
- \(L(M_3) = \{a\} = L(a)\)

### Inductive Hypothesis

Assume for regular expressions \(r_1\) and \(r_2\) that \(L(r_1)\) and \(L(r_2)\) are regular languages

### Inductive Step

We will prove:

- \(L(r_1 + r_2)\) \(\subseteq\) regular languages
- \(L(r_1 \cdot r_2)\) \(\subseteq\) regular languages
- \(L(r_1^*)\) \(\subseteq\) regular languages
- \(L((r_1))\) \(\subseteq\) regular languages

By definition of regular expressions:

- \(L(r_1 + r_2) = L(r_1) \cup L(r_2)\)
- \(L(r_1 \cdot r_2) = L(r_1)L(r_2)\)
- \(L(r_1^*) = (L(r_1))^*\)
- \(L((r_1)) = L(r_1)\)

By inductive hypothesis:

- \(L(r_1)\) and \(L(r_2)\) are regular languages

We also know:

- Regular languages are closed under union
- \(L(r_1) \cup L(r_2)\)
- Regular languages are closed under concatenation
- \(L(r_1)L(r_2)\)
- Regular languages are closed under star
- \((L(r_1))^*\)
Therefore
\[ L(r_1 + r_2) = L(r_1) \cup L(r_2) \]
\[ L(r_1 \cdot r_2) = L(r_1)L(r_2) \]
\[ L(r_1^*) = (L(r_1))^* \]

And trivially \( L((r_1)^*) \) is a regular language.

Proof – Part 2
For any regular language \( L \) there is a regular expression \( r \) such that \( L(r) = L \)
Proof by construction of regular expression

Since \( L \) is regular
take the NFA \( M \) that accepts it

The NFA can then be reduced to a regular expression by the state elimination algorithm.

\[ L(M) = L \]

We are going to describe the state elimination algorithm later on in this lecture.

From \( M \) construct the equivalent
Generalized Transition Graph (a graph in which transition labels are regular expressions)

Example

\[ a \]
\[ a, b \]
\[ a+b \]

Reverse of a Regular Language

Theorem
The reverse \( L^R \) of a regular language \( L \) is a regular language

Proof idea
Construct NFA that accepts \( L^R \):
invert the transitions of the NFA that accepts \( L \)

Since \( L \) is regular,
there is a NFA that accepts \( L \)

Example

\[ L = ab^* + ba \]

Invert Transitions

Make old initial state a final state

Add a new initial state
Resulting machine accepts $L^R$ is regular

$L = ab^* + ba$

$L^R = b^*a + ab$

---

**Minimizing DFA**

The deterministic finite automata are not always the smallest possible accepting the source language.

There may be states with the same "acceptance behavior". This applies to states p and q, if for all input words, the automaton always or never moves to a final state from p and q.

---

**State Reduction by Set Partitioning**

The set partitioning technique is similar to one used for partitioning people into groups based on their responses to questionnaire.

The following slides show the detailed steps for computing equivalent state sets of the starting DFA and constructing the corresponding reduced DFA.

**Step 0:** Partition the states into two groups accepting and non-accepting.

<table>
<thead>
<tr>
<th>$P_1$</th>
<th>$P_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ 3, 4, 5 }</td>
<td>{ 0, 1, 2 }</td>
</tr>
</tbody>
</table>

**Step 1:** Get the response of each state for each input symbol. Notice that States 3 and 0 show different responses from the ones of the other states in the same set.

**Step 2:** Partition the sets according to the responses, and go to Step 1 until no partition occurs.

<table>
<thead>
<tr>
<th>$P_{11}$</th>
<th>$P_{12}$</th>
<th>$P_{21}$</th>
<th>$P_{22}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{11}$</td>
<td>$p_{12}$</td>
<td>$p_{21}$</td>
<td>$p_{22}$</td>
</tr>
<tr>
<td>a→↑↑↑↑↑↑</td>
<td>a→↑↑↑↑↑↑</td>
<td>a→↑↑↑↑↑↑</td>
<td>a→↑↑↑↑↑↑</td>
</tr>
<tr>
<td>(4, 5)</td>
<td>(3)</td>
<td>(1, 2)</td>
<td>(0)</td>
</tr>
<tr>
<td>b→↓↓↓↓↓↓</td>
<td>b→↓↓↓↓↓↓</td>
<td>b→↓↓↓↓↓↓</td>
<td>b→↓↓↓↓↓↓</td>
</tr>
<tr>
<td>$p_{11}$</td>
<td>$p_{11}$</td>
<td>$p_{11}$</td>
<td>$p_{11}$</td>
</tr>
<tr>
<td>$p_{11}$</td>
<td>$p_{11}$</td>
<td>$p_{11}$</td>
<td>$p_{11}$</td>
</tr>
</tbody>
</table>

No further partition is possible for the sets $P_{11}$ and $P_{21}$. So the final partition results are as follows:

\{ 4, 5 \} \{ 3 \} \{ 1, 2 \} \{ 0 \}

Minimized DFA consists of four states of the final partition, and the transitions are the one corresponding to the starting DFA.

---

Minimized DFA

Starting DFA
DFA Minimization Algorithm

The algorithm

\[ P \leftarrow \{ F, \{Q-F}\} \]

while ( \( P \) is still changing)

\[ T \leftarrow \{ \} \]

for each set \( s \in P \)

for each \( \alpha \in \Sigma \)

partition \( s \) by \( \alpha \) into \( s_1, s_2, \ldots, s_k \)

\[ T \leftarrow T \cup s_1, s_2, \ldots, s_k \]

if \( T \neq P \) then

\[ P \leftarrow T \]

Why does this work?

Partition \( P \in 2^Q \) (power set)

Start off with 2 subsets of \( Q \)

(\{F\} and \(Q-F\))

While loop takes \( P \rightarrow P \) by splitting one or more sets

\( P_{n+1} \) is at least one step closer to the partition with \( |Q| \) sets

Maximum of \( |Q| \) splits

Partitions are never combined

Initial partition ensures that final states are intact

This is a fixed-point algorithm!

Converting FA to a Regular Expression

Add a new start (s) and a new final (f) state:

- From s to the ex-starting state (1)
- From each ex-final state (1,3) to f

Let's remove state 1!

Each combination of input/output to 1 will generate a new path once state 1 is gone

\[ \lambda : a, b \]

\[ \lambda : \lambda \]

\[ a : a, b \]

\[ a : \lambda \]

When state 1 is gone we must be able to make all those transitions!

\[ \lambda : a, b \]

\[ \lambda (a \cup b) \]

Previous:

\[ \lambda : \lambda \]

\[ \lambda (a \cup b) \]

A common mistake: having several arrows between the same pair of states. Join the two arrows (by union of their regular expressions) before going on to the next step.
Union again...

Without state 1...

Now we repeat the same procedure for the state 2...

Following the path s-2-3 we concatenate all strings on our way... don't forget a* in 2!

When 2 is removed, the path 3-2-3 has also to be preserved, so we concatenate 3-2 string, take a* loop and go back 2-3 with string b!

This is how the FA looks like without state 2:

Finally we remove state 3...

...so we concatenate strings s-3, loop in 3 and 3-f

Now we can omit state 3!

From s we have two choices, empty string or the long expression OR is represented by union ∪, as usually
So union the arrows...

\[(a \cup b)a'b \cup ((a(a \cup b)b)a'b)^* (\lambda \cup a) \cup \lambda\]

...and we are done!

Converting FA to a Regular Expression: An Algorithm

- We expand our notion of NFA- to allow transitions on arbitrary regular expressions, not simply single symbols or \( \lambda \).
- Successively eliminate states, replacing transitions that enter and leave a state with a more complicated regular expression, until eventually there are only two states left: a start state, an accepting state, and a single transition connecting them, labeled with a regular expression.
- The resulting regular expression is then our answer.

To begin with, the automaton should have a start state that has no transitions into it (including self-loops), and which is not accepting.
- If your automaton does not obey this property, add a new, non-accepting start state \( s \), and add a \( \lambda \)-transition from \( s \) to the original start state.
- The automaton should also have a single accepting final state with no transitions leaving it, and no self-loops.
- If your automaton does not have it, add a new final state \( q \), change all other accepting states to non-accepting, and add \( \lambda \)-transitions from them to \( q \).

This change clearly doesn't change the language accepted by the automaton.

Repeat the following steps, which eliminate a state:
1. Pick a non-start, non-accepting state \( q \) to eliminate.
   The state \( q \) will have \( i \) transitions in and \( j \) transitions out. Each will be labelled with a regular expression.
   For each of the \( ij \) combinations of transitions into and out of \( q \), replace:

   \[ A \setminus B \]

   with

   \[ A \cup B \]

   And delete state \( q \).

2. If several transitions go between a pair of states, replace them with a single transition that is the union of the individual transitions.
   E.g. replace:

   \[ A \]

   with

   \[ A \cup B \]

Example

\[ (b \cup ab^*a)^* \]

\[ (b \cup ab^*a)b \]

\[ a^* \]

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

N.B.

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

used on s 46 and 47 in Sallings book