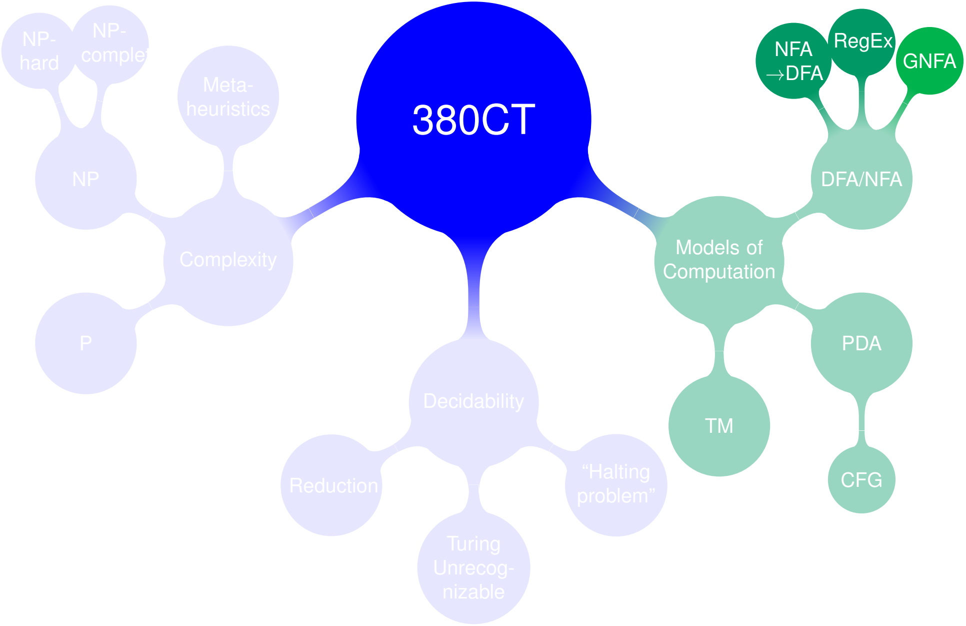
Models of Computation: NFA $\longleftrightarrow$ DFA Regular Expressions

Dr Kamal Bentahar

School of Computing, Electronics and Mathematics
Coventry University

Week 3 – 5/2/2018

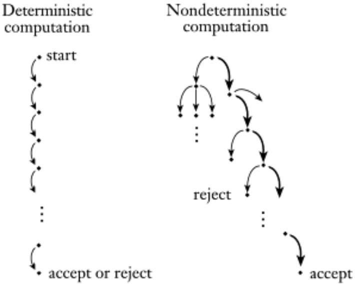
Mindmap
NFA $\longrightarrow$ DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx $\longrightarrow$ NFA
NFA $\longrightarrow$ RegEx
GNFA
NFA $\longrightarrow$ GNFA
GNFA $\longrightarrow$ RegEx
Summary



Mindmap
NFA $\longrightarrow$ DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx $\longrightarrow$ NFA
NFA $\longrightarrow$ RegEx
GNFA
NFA $\longrightarrow$ GNFA
GNFA $\longrightarrow$ RegEx
Summary

Last time: DFAs & NFAs

- **DFA**: $\delta : Q \times \Sigma \to Q$
- **NFA**: $\delta : Q \times \Sigma \to 2^Q$



Surprising result

NFAs recognize exactly the same languages as DFAs.

Observation: DFAs are a *special case* of NFAs. For example:

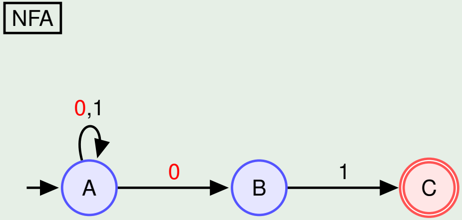
DFA	a	b
$\to A$	A	B
$*B$	A	B

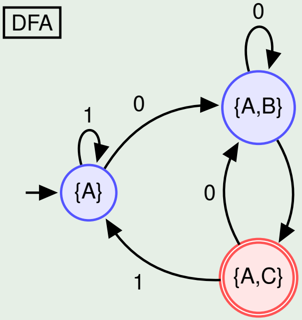
$\to$

NFA	a	b
$\to A$	$\{A\}$	$\{B\}$
$*B$	$\{A\}$	$\{B\}$

How about the reverse?
Can we convert any NFA into a DFA?

Example (The **Subset construction method**)





DFA	0	1
→ {A}	{A,B}	{A}
{A,B}	{A,B}	{A,C}
* {A,C}	{A,B}	{A}

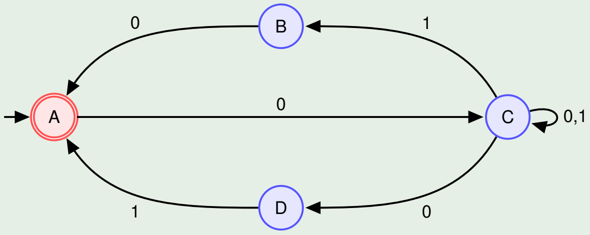
Example (The subset construction method directly applied to a table)

NFA	0	1
A	{A, B}	{A, B}
* B	{A}	{C}
→ C	{A}	{A}

$\rightarrow$

DFA	0	1
→ {C}	{A}	{A}
{A}	{A, B}	{A, B}
* {A,B}	{A,B}	{A,B,C}
* {A,B,C}	{A,B}	{A,B,C}

Example (A longer example)



NFA	0	1
→* A	{C}	∅
B	{A}	∅
C	{C,D}	{C,B}
D	∅	{A}

→

DFA	0	1
→* {A}	{C}	∅
{C}	{C,D}	{C,B}
{C,D}	{C,D}	{C,B,A}
{C,B}	{C,D,A}	{C,B}
* {C,B,A}	{C,D,A}	{C,B}
* {C,D,A}	{C,D}	{C,B,A}
∅	∅	∅

Mindmap
NFA → DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx → NFA
NFA → RegEx
GNFA
NFA → GNFA
GNFA → RegEx
Summary

The subset construction method

Given an NFA $N = (Q, \Sigma, \delta, q_{\text{start}}, F)$, we can construction an equivalent DFA $D = (Q', \Sigma, \delta', \{q_{\text{start}}\}, F')$ as follows:

- $Q' \subset 2^Q$ is the set of all possible states that can be reached from q_{start}.
- For each entry $(A, s) \in Q' \times \Sigma$ in the transition table of D, we find the result $\delta'(q, s)$ as the **union** of all $\delta(q, s)$ for all $q \in A$
- $F' \subset Q'$ contains all the sets that have a state from F.

Regular Languages

Mindmap
NFA ⟶ DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx ⟶ NFA
NFA ⟶ RegEx
GNFA
NFA ⟶ GNFA
GNFA ⟶ RegEx
Summary

Theorem: The equivalence of NFAs and DFAs

Every NFA has an equivalent DFA.

Theorem: NFAs and DFAs recognize the same languages

NFAs and DFAs are equivalent in terms of languages recognition.

Definition (Regular Languages)

A language is **regular** if and only if some NFA recognizes it.

Extension: ε-NFAs $\longleftrightarrow$ Regular Languages

We allow ε as a transition label.

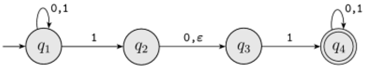
Definition of ε-NFAs

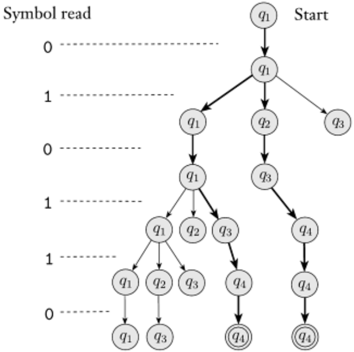
An ε-NFA is defined by the 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, F)$ like normal NFAs, but where the transition function is given by

$$\delta \colon Q \times \Sigma_\varepsilon \to 2^Q \quad \text{where } \Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}.$$

Definition (Regular Languages)

A language is **regular** if and only if some ε-NFA recognizes it.





Mindmap
NFA $\to$ DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx $\longrightarrow$ NFA
NFA $\longrightarrow$ RegEx
GNFA
NFA $\longrightarrow$ GNFA
GNFA $\longrightarrow$ RegEx
Summary

Regular operations

Let A and B be two languages.
The following operations are called **the regular operations**:

1. **Union:** $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
   i.e. strings from A <u>or</u> from B.

2. **Concatenation:** $AB = \{xy \mid x \in A \text{ and } y \in B\}$
   i.e. string from A <u>followed</u> by string from B.

3. **Star:** $A^* = \{x_1 x_2 \cdots x_n \mid n \geq 0 \text{ and each } x_i \in A\}$
   i.e. concatenations of <u>zero or more</u> strings from A.

$$A^* = \{\varepsilon\} \cup A \cup AA \cup AAA \cup \cdots = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \cdots$$

NFA $\leftrightarrow$ DFA $\leftrightarrow$ RegEx

Mindmap
NFA $\longrightarrow$ DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx $\longrightarrow$ NFA
NFA $\longrightarrow$ RegEx
GNFA
NFA $\longrightarrow$ GNFA
GNFA $\longrightarrow$ RegEx
Summary

Regular Languages – closures

If L and M are two regular languages then the following are also regular

1. $L \cup M$ (Union: string in L or M)
2. LM (Concatenation: string from L followed by string M)
3. L^* (Star: $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$)

Theorem

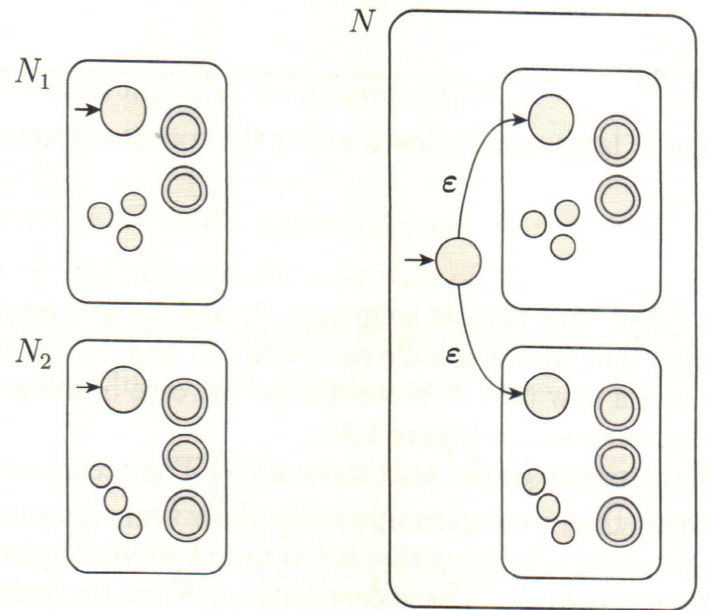
The class of regular languages is closed under the regular operations (union, concatenation, and star).

Proof: Next 3 slides.

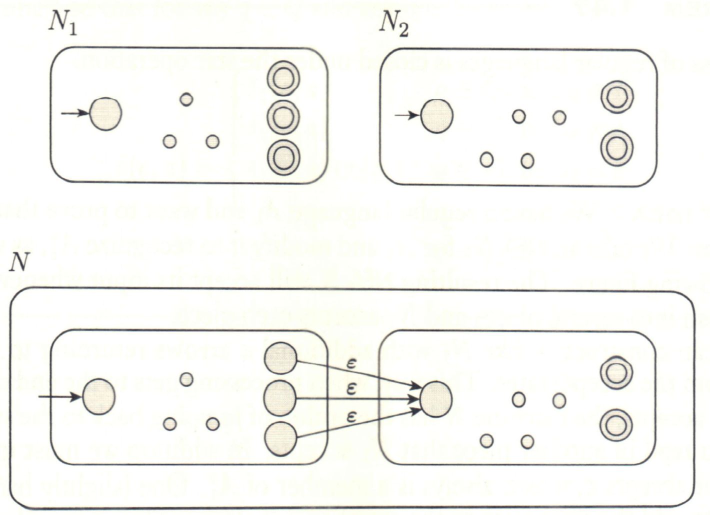
NFA ↔ DFA ↔ RegEx

Mindmap
NFA → DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx → NFA
NFA → RegEx
GNFA
NFA → GNFA
GNFA → RegEx
Summary

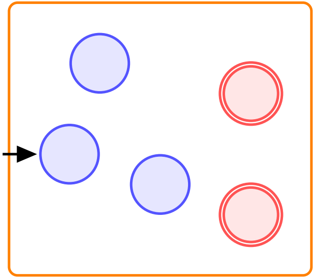
Proof: Closure under Union

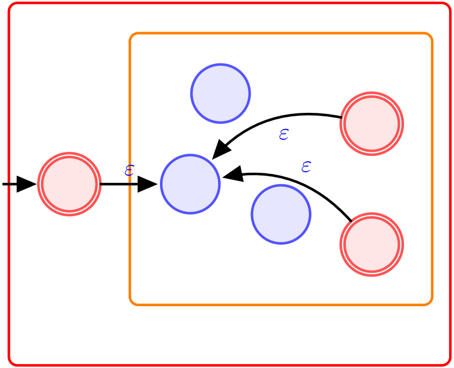


Proof: Closure under Concatenation



Proof: Closure under Star





Mindmap
NFA → DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx → NFA
NFA → RegEx
GNFA
NFA → GNFA
GNFA → RegEx
Summary

Regular expressions

We can describe NFAs using **Finite Automata.**

We can also describe them using **Regular Expressions.**

Example

Let $\Sigma = \{0, 1\}$

- The finite language $\{1, 11, 00\}$: $1 + 11 + 00$
- Strings ending with 0: Σ^*0
- Strings starting with 11: $11\Sigma^*$
- Strings of even length: $(\Sigma\Sigma)^*$

Definition (Regular Expressions – Recursive definition)

R is said to be a regular expression (RegEx) if and only if

- R is $\emptyset$ or ε or a single symbol from the alphabet
- or R is the union, concatenation or star of other (“smaller”) RegEx’s.

NFA ↔ DFA ↔ RegEx

Mindmap

NFA → DFA

Regularity

ε-NFAs

Regular operations

Regular expressions

RegEx → NFA

NFA → RegEx

GNFA

NFA → GNFA

GNFA → RegEx

Summary

Regular Languages $\longleftrightarrow$ Regular Expressions

Notation for writing RegEx's:

- **Union:** $+$
- **Concatenation:** Juxtaposition (i.e. no symbol)
- **Star:** $*$ as a superscript

Unless brackets are used to explicitly denote precedence, the **operators precedence** for the regular operations is: star, concatenation, then union.

Theorem

A language is regular if and only if some regular expression describes it.

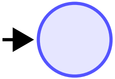
Constructive proof in two parts:

- (1/2): RegEx $\rightarrow$ NFA
- (2/2): NFA $\rightarrow$ RegEx

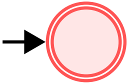
Proof (1/2): RegEx $\rightarrow$ NFA

We cover all the possible cases from the definition of RegEx's:

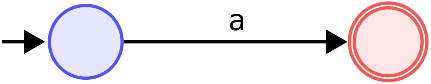
1. $R = \emptyset$



2. $R = \varepsilon$



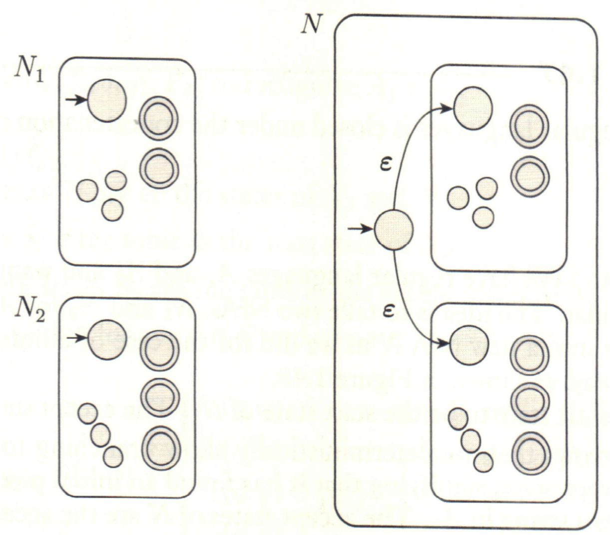
3. $R = \mathsf{a} \in \Sigma$



Mindmap
NFA $\rightarrow$ DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx $\longrightarrow$ NFA
NFA $\longrightarrow$ RegEx
GNFA
NFA $\longrightarrow$ GNFA
GNFA $\longrightarrow$ RegEx
Summary

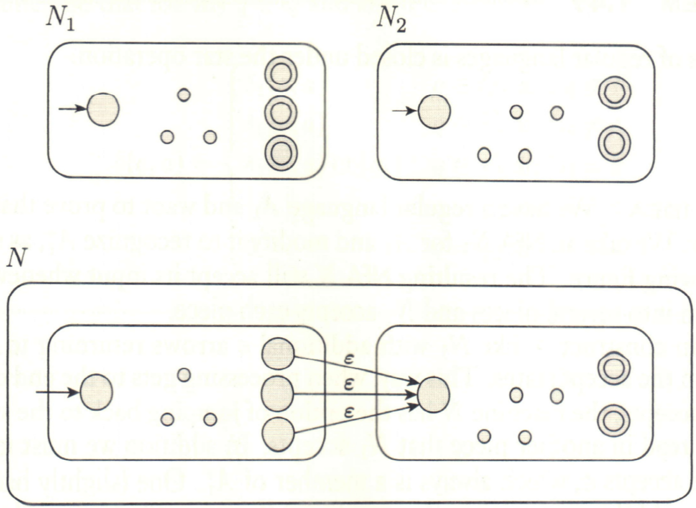
Proof (1/2): RegEx $\rightarrow$ NFA

4. $R = A + B$ (Union)



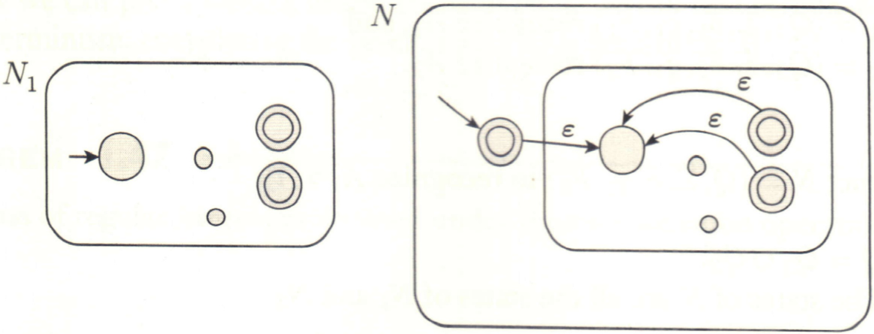
Proof (1/2): RegEx $\rightarrow$ NFA

5. $R = AB$ (Concatenation)



Proof (1/2): RegEx $\to$ NFA

6. $R = A^*$ (Star)



Mindmap
NFA $\to$ DFA
Regularity
ε-NFAs
Regular operations
Regular expressions
RegEx $\to$ NFA
NFA $\to$ RegEx
GNFA
NFA $\to$ GNFA
GNFA $\to$ RegEx
Summary

Proof (2/2): NFA $\to$ RegEx

We introduce a machine to help us produce RegEx's for any given NFA:

Generalized Nondeterministic Finite Automaton (GNFA)

GNFAs are similar to NFAs but have the following restrictions/extensions:

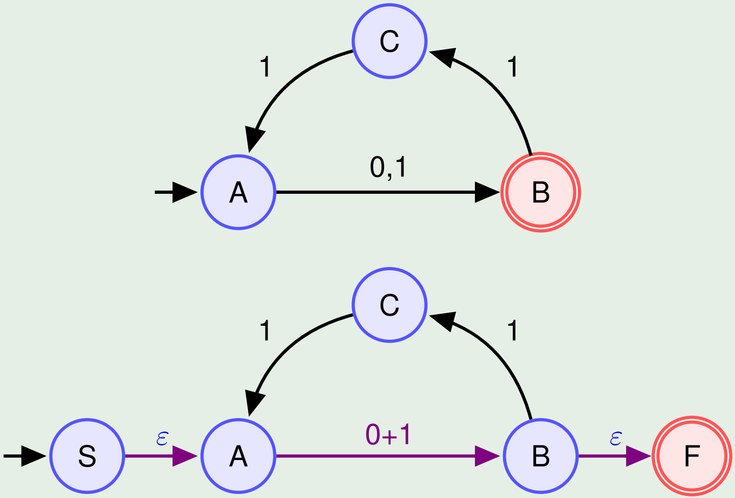
1. Only **one accept state**
2. **Initial state:** no in-coming transitions
3. **Accept state:** no out-going transitions
4. **Transitions:** RegEx's, rather than just symbols from the alphabet

We can convert any NFA into a GNFA in three steps:

1. Add a **new start state** with an ε-transition to the NFA's start state.
2. Add a **new accept state** with ε-transitions from the NFA's accept states.
3. Replace **transitions that have multiple labels** with their union. (e.g. $a, b \to a + b$.)

Proof (2/2): NFA $\rightarrow$ RegEx | Converting NFA into GNFAs

Example (NFA $\rightarrow$ GNFA)



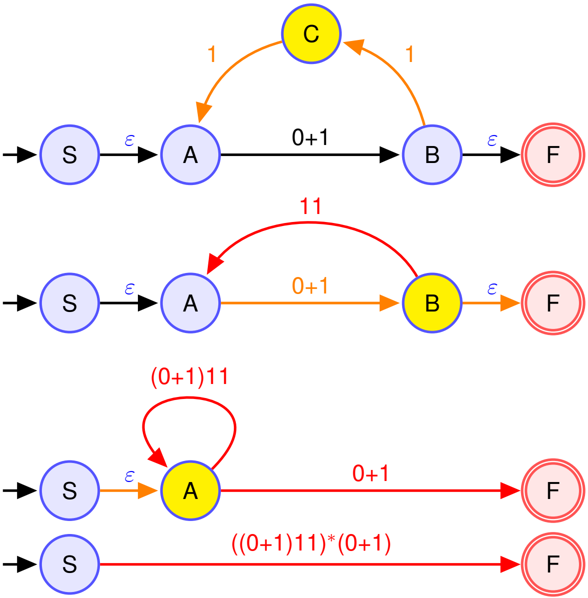
Proof (2/2): NFA $\rightarrow$ RegEx | Reducing GNFAs into RegEx's

Key observation: Given a GNFA, the "inner states" may be removed from it, one at a time, with regular expressions replacing each removed transition. We end with only the initial and accept states, and a single transition between them, labelled with a regular expression.

The GNFA Algorithm

1. Convert the NFA to a GNFA.
2. Remove the "inner states," one at a time, and replace the affected transitions using RegEx's.
3. Repeat until only two states (initial and accept) remain.
4. The RegEx on the only remaining transition is the required RegEx.

Example



C
1
1
S
ε
A
0+1
B
ε
F
11
S
ε
A
0+1
B
ε
F
(0+1)11
S
ε
A
0+1
F
S
((0+1)11)*(0+1)
F

Summary

- Introduced GNFAs as a means of converting NFAs to equivalent RegEx's
- Demonstrated how to turn an NFA into a GNFA
- Demonstrated how to obtain RegEx's from a GNFA by removing states one at a time
- The set of regular languages is exactly equal to the set of languages described by some RegEx/GNFA/ε-NFA/NFA/DFA.

Regular Languages

The class of regular languages can be:

1. Recognized by NFAs. (equiv. GNFA or ε-NFA or NFA or DFA).
2. Described using **Regular Expressions.**
3. Generated using **Linear Grammars.** (See this later!)