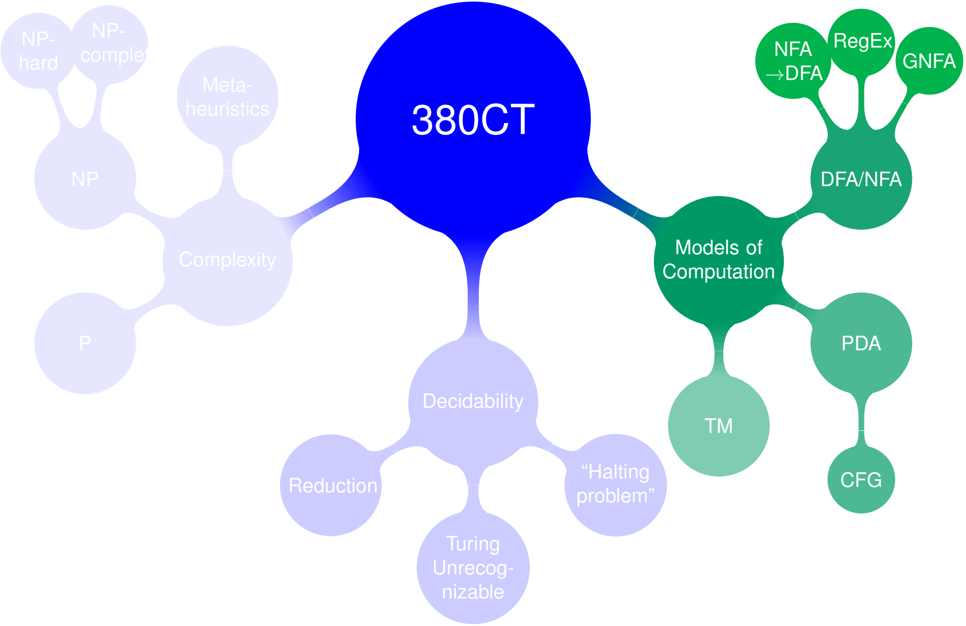
Models of Computation: NFA $\Longleftrightarrow$ DFA & Regular Expressions

Dr Kamal Bentahar

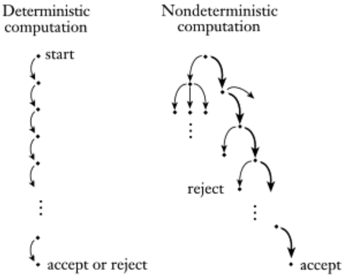
School of Engineering, Environment and Computing
Coventry University

17/10/2016


380CT
NP-hard
NP-comple
Meta-heuristics
NP
Complexity
P
Decidability
Reduction
Turing Unrecog-nizable
“Halting problem”
Models of Computation
DFA/NFA
NFA →DFA
RegEx
GNFA
PDA
CFG
TM

Last time: DFAs & NFAs

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



Surprise: NFAs recognize exactly the same languages as DFAs!

Note: DFAs are a *special case* of NFAs, e.g. given a DFA defined by

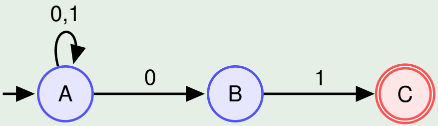
	a	b
$\to A$	A	B
$*B$	A	B

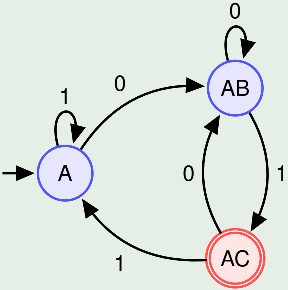
$\to$

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

How about the reverse?
Can we show that any NFA can be converted into a DFA?

Example (Subset construction method)



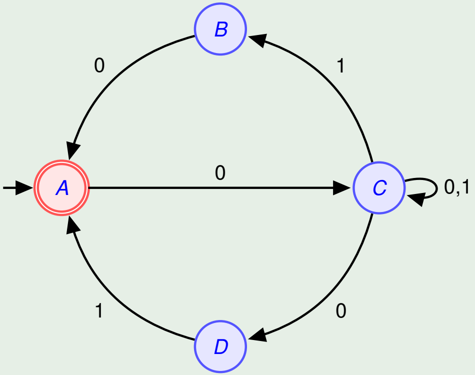


States	0	1
$\rightarrow\{A\}$	$\{A, B\}$	$\{A\}$
$\{A, B\}$	$\{A, B\}$	$\{A, C\}$
$*\{A, C\}$	$\{A, B\}$	$\{A\}$

$\Longleftrightarrow$

States	0	1
$\rightarrow A$	AB	A
AB	AB	AC
$*AC$	AB	A

Example (Subset construction method)



	0	1
$\rightarrow * \{A\}$	$\{C\}$	$\emptyset$
$\{C\}$	$\{C, D\}$	$\{C, B\}$
$\{C, B\}$	$\{C, D, A\}$	$\{C, B\}$
$* \{C, B, A\}$	$\{C, D, A\}$	$\{C, B\}$
$\{C, D\}$	$\{C, D\}$	$\{C, B, A\}$
$* \{C, D, A\}$	$\{C, D\}$	$\{C, B, A\}$
$\emptyset$	$\emptyset$	$\emptyset$

Regular Languages

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.

ε-NFAs $\longleftrightarrow$ Regular Languages

Definition of ε-NFAs

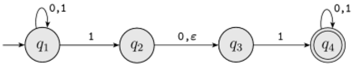
An ε-NFA is defined by the 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, F)$ where Q is the set of states, Σ is the alphabet, q_{start} is the start state, F is the set of accepting states, and

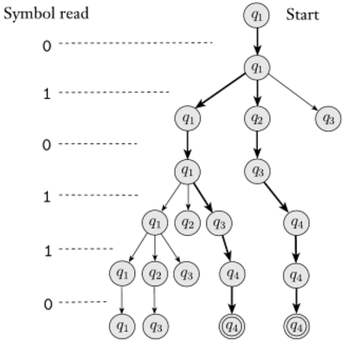
$$\delta\colon Q \times \Sigma_\varepsilon \to 2^Q \quad \text{where } \Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$$

is the transition function (which can be partial).

Definition (Regular Languages)

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





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 or from B.

2. **Concatenation:** $AB = \{xy \mid x \in A \text{ and } y \in B\}$
   i.e. string from A followed 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 zero or more strings from A.

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

Regular Languages – closures

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

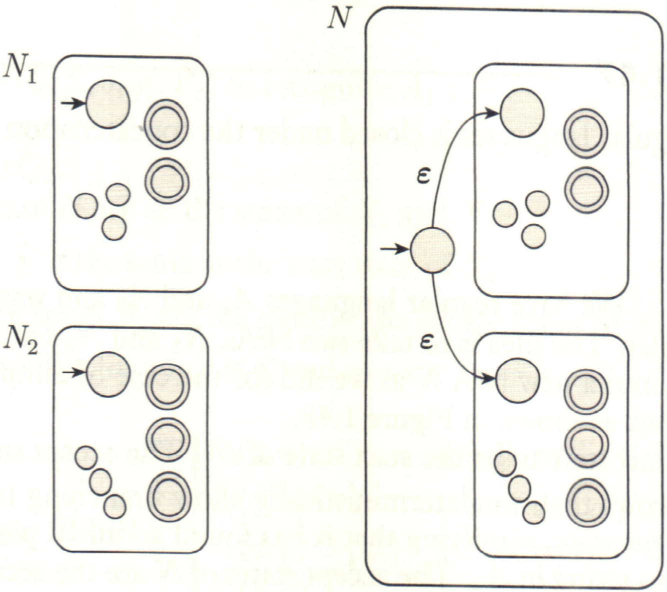
1. $L \cup M$ (Union: string in L or M)
2. LM (Concatenation: string from L followed by string M)
3. L^* (Star: $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$)

Theorem

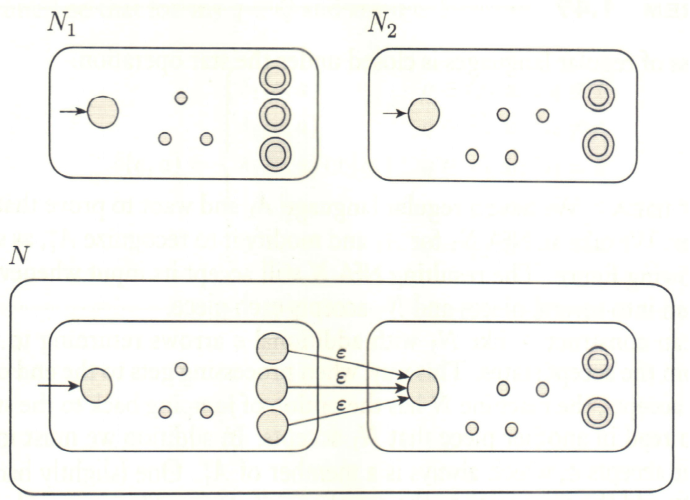
The class of regular languages is closed under the regular operations (union, concatenation, and star).

Proof: Next 3 slides.

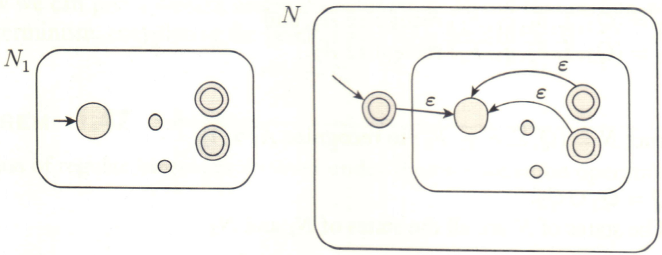
Proof: Closure under Union



Proof: Closure under Concatenation



Proof: Closure under Star



Regular expressions

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.

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: $(\Sigma\Sigma)^*$

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.

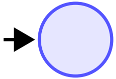
Constructive proof in two parts:

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

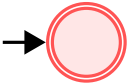
Proof (1/2): RegEx $\rightarrow$ NFA

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

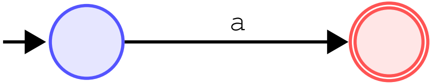
1. $R = \emptyset$



2. $R = \varepsilon$

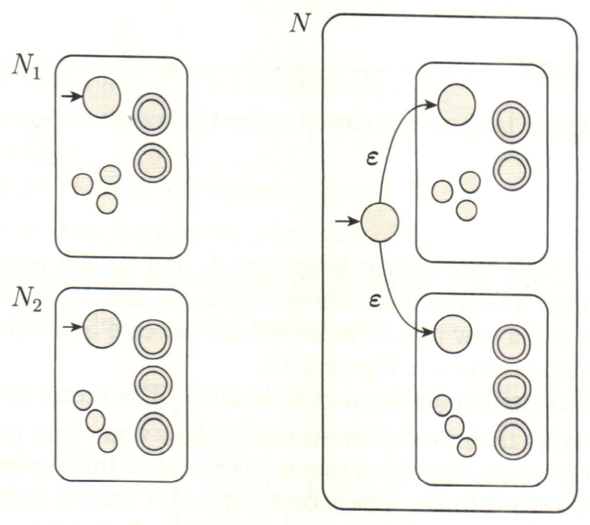


3. $R = \mathtt{a} \in \Sigma$



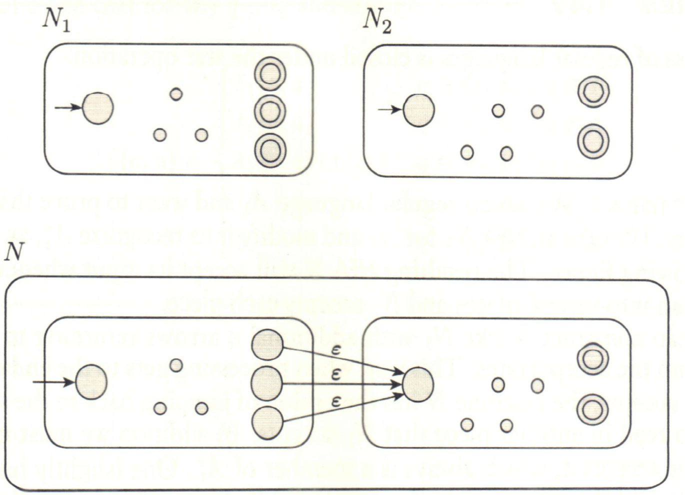
Proof (1/2): RegEx $\rightarrow$ NFA

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



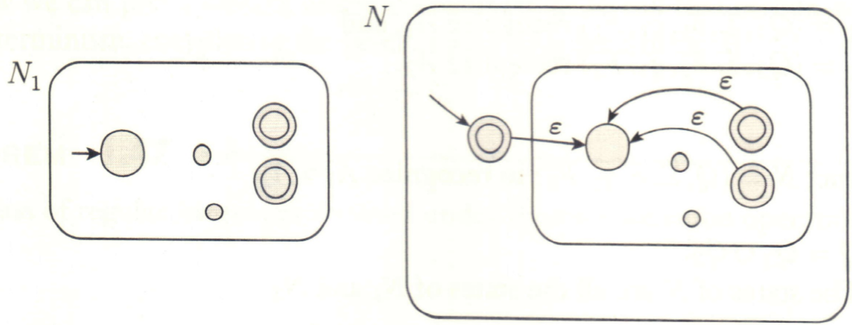
Proof (1/2): RegEx $\to$ NFA

5. $R = AB$ (Concatenation)



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

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



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

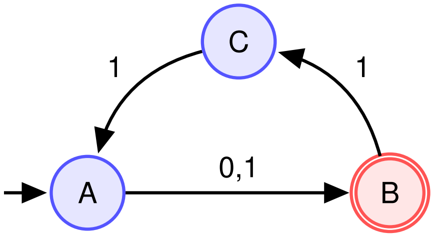
We can convert any NFA into a GNFA as follows:

- Add a new start state with an ε-transition to the NFA's start state.
- Add a new accept state with ε-transitions from the NFA's accept states.
- If a transition has multiple labels then replace them with their union. (e.g. $a, b \rightarrow a + b$.)

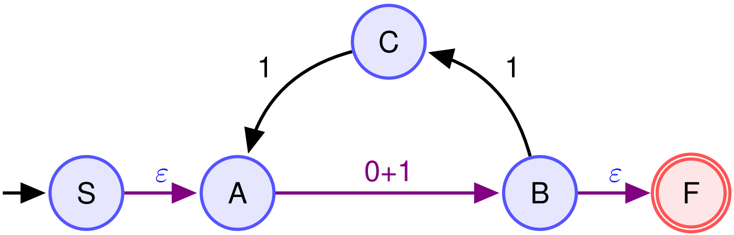
Proof (2/2): NFA $\rightarrow$ RegEx | Converting NFA into GNFAs

Example

NFA:



GNFA:



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

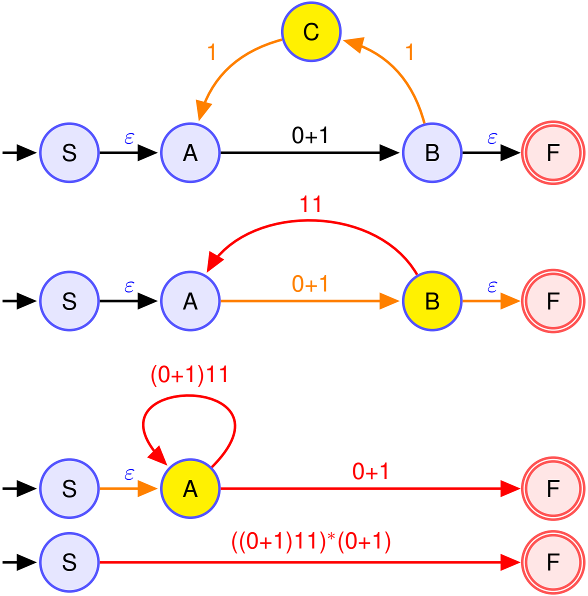
Key observation: Once the GNFA is produced, the original NFA states may be removed from it, one at a time, with the regular expressions redefined for each removed transition.

Eventually, we end up with only two states (initial and accept) and a single transition between them, labelled with a regular expression – this is the equivalent RegEx for the NFA.

The GNFA Algorithm

1. Convert the NFA to a GNFA.
2. Remove states, one at a time, and "patch" any 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 equivalent RegEx to the NFA.

Example



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!)