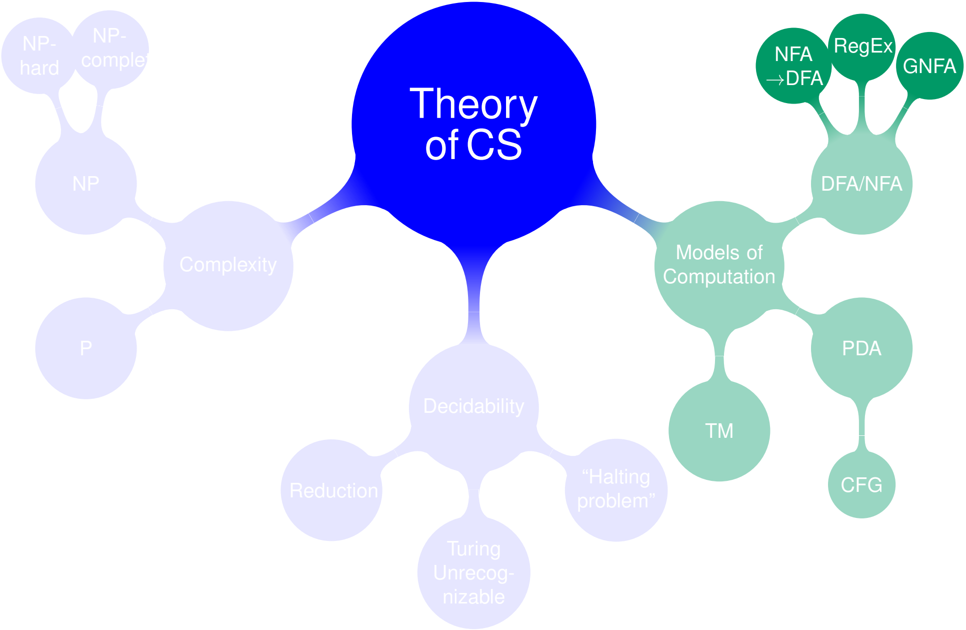
Models of Computation:
DFA $\leftrightarrow$ NFA $\leftrightarrow$ Regular Expressions

Dr Kamal Bentahar

School of Computing, Electronics and Mathematics
Coventry University

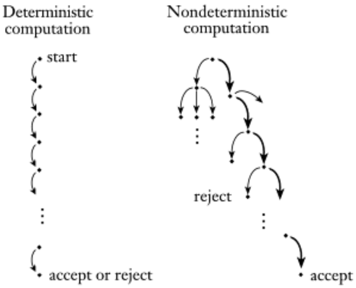
Lecture 3


Theory of CS
Complexity
NP
NP-hard
NP-complete
P
Decidability
Reduction
Turing Unrecognizable
"Halting problem"
Models of Computation
DFA/NFA
NFA →DFA
RegEx
GNFA
PDA
CFG
TM

Last time: DFAs & NFAs

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

Computation schematic:



Surprising result

NFAs recognize exactly the same languages as DFAs.

Image of a function

The set of "all the values taken by δ" is called the **image** of δ.

Example

If $Q = \{A, B, C\}$ and δ is given by

	0	1
$\rightarrow A$	B	B
$*B$	B	C
C	C	C

then the image of δ is $\{B, C\}$, which is a subset of Q.

1/2) DFA $\rightarrow$ NFA

- Given a DFA, how do we construct an equivalent NFA to it?

Observation: DFAs are a *special case* of NFAs!
Technically, we interpret each state q from the image of δ as a set $\{q\}$.

Example

DFA	0	1
$\rightarrow A$	B	B
$*B$	B	C
C	C	C

$\rightarrow$

NFA	0	1
$\rightarrow A$	$\{B\}$	$\{B\}$
$*B$	$\{B\}$	$\{C\}$
C	$\{C\}$	$\{C\}$

2/2) DFA $\leftarrow$ NFA

▶ How about the reverse? Can we convert any NFA to an equivalent DFA that recognizes the same language?

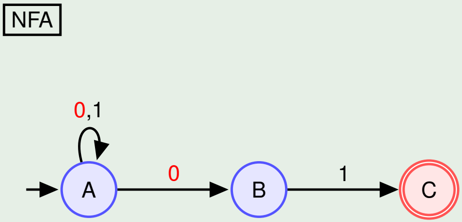
Idea: Build a DFA that simulates how the NFA works.

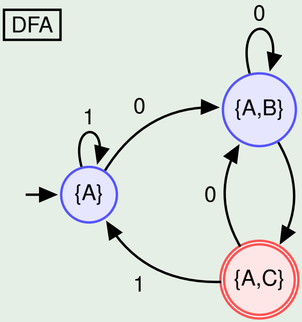
- All we need to keep track of is the **current set of states** used by the NFA.
- If n is the number of states of the NFA then there are 2^n subsets of states.
- Each subset corresponds to a possibility that the DFA must remember.

Let us see some examples...

2/2) DFA ← NFA

Example (The **Subset construction method**)





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

2/2) DFA $\leftarrow$ NFA

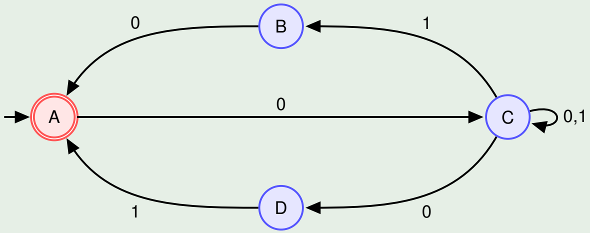
Example (The subset construction method directly applied to a table)

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

$\rightarrow$

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

$\rightarrow$

DFA	0	1
$\rightarrow$* {A}	{C}	$\emptyset$
{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}
$\emptyset$	$\emptyset$	$\emptyset$

2/2) NFA $\to$ DFA (The subset construction method)

The subset construction method

Given an NFA $N = (Q, \Sigma, \delta, q_{\text{start}}, F)$, we can construct 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 $\delta'(A, s)$ as the **union** of all $\delta(q, s)$ for all $q \in A$, i.e.

$$\delta'(A, s) = \bigcup_{q \in A} \delta(q, s)$$

- $F' \subset Q'$ contains all the sets that have a state from F.

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.

Extension: ε-NFAs $\longleftrightarrow$ Regular Languages

We allow ε as a transition label.

Definition of ε-NFAs

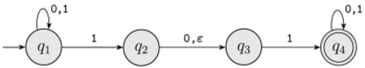
An ε-NFA is defined by the 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, F)$ like normal NFAs, but where the transition function is given by

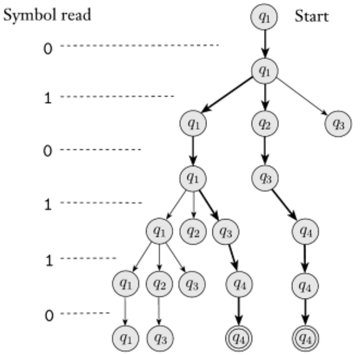
$$\delta\colon Q \times \Sigma_\varepsilon \to 2^Q \quad \text{where } \Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}.$$

These can also be converted to NFAs using the subset construction method. So we can also say:

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 – “Closure” under the regular operations

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

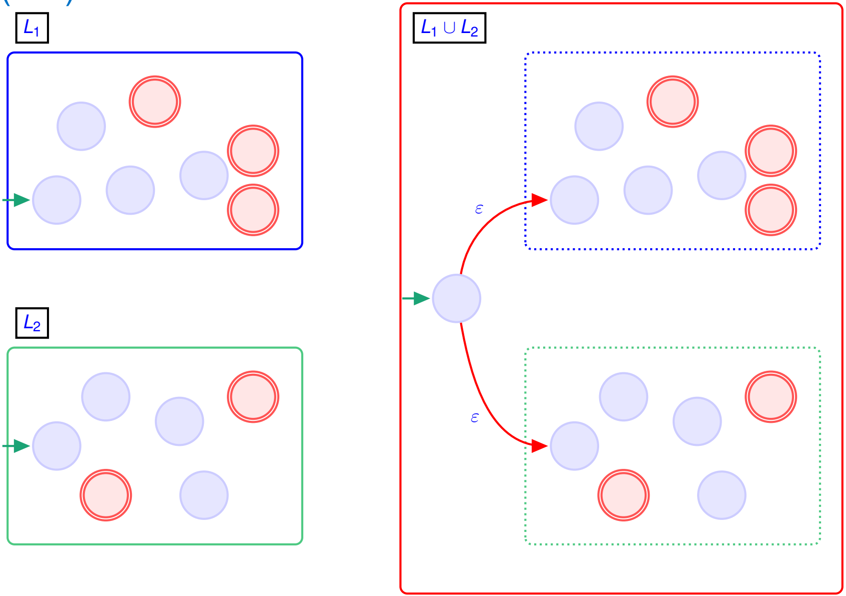
1. $L \cup M$ (Union: string in L or M)
2. LM (Concatenation: string from L followed by string M)
3. L^* (Star: $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$)

Theorem

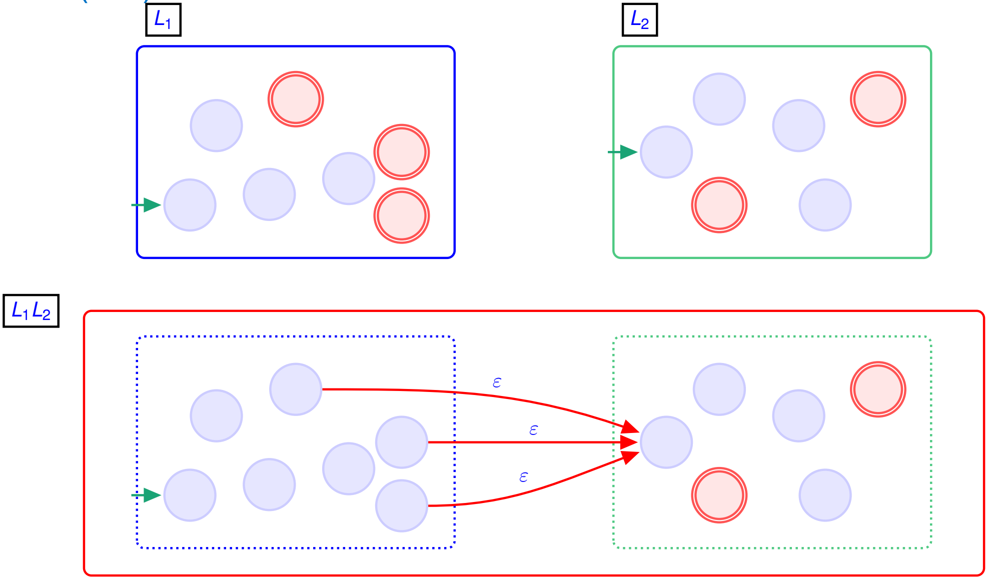
The class of regular languages is closed under the regular operations (union, concatenation, and star).

Proof outline: Next 3 slides.

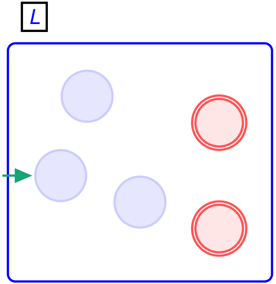
Proof (1/3): Closure under Union

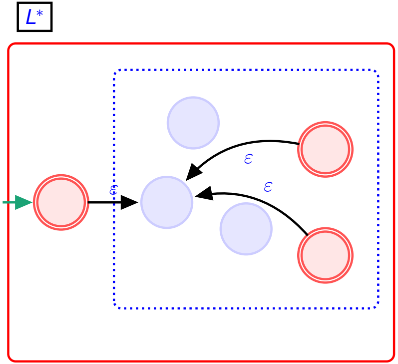


Proof (2/3): Closure under Concatenation



Proof (3/3): Closure under Star





Regular Expressions

We can describe NFAs using **Finite Automata** (Accept/Reject strings).
We can also describe them using **Regular Expressions** (Generate strings).

Example

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

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

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.

Regular Languages $\longleftrightarrow$ Regular Expressions

Notation for writing RegEx's:

- **Union:** Plus: $\blacksquare + \blacksquare$ (Textbook uses $\blacksquare \cup \blacksquare$)
- **Concatenation:** Juxtaposition: $\blacksquare\blacksquare$ (i.e. no symbol)
- **Star:** $*$ as a superscript: $\blacksquare^*$

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:

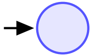
- (1/2): RegEx $\rightarrow$ NFA
- (2/2): NFA $\rightarrow$ RegEx

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

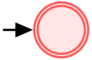
We need to cover all the 6 possible cases from the definition of RegEx's:

Base cases:

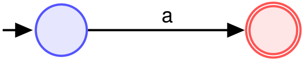
1. $R = \emptyset$



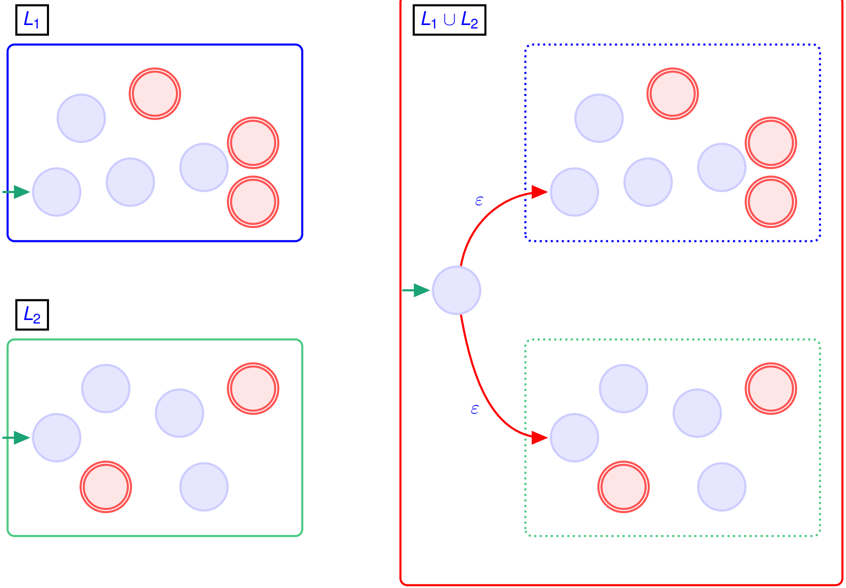
2. $R = \varepsilon$



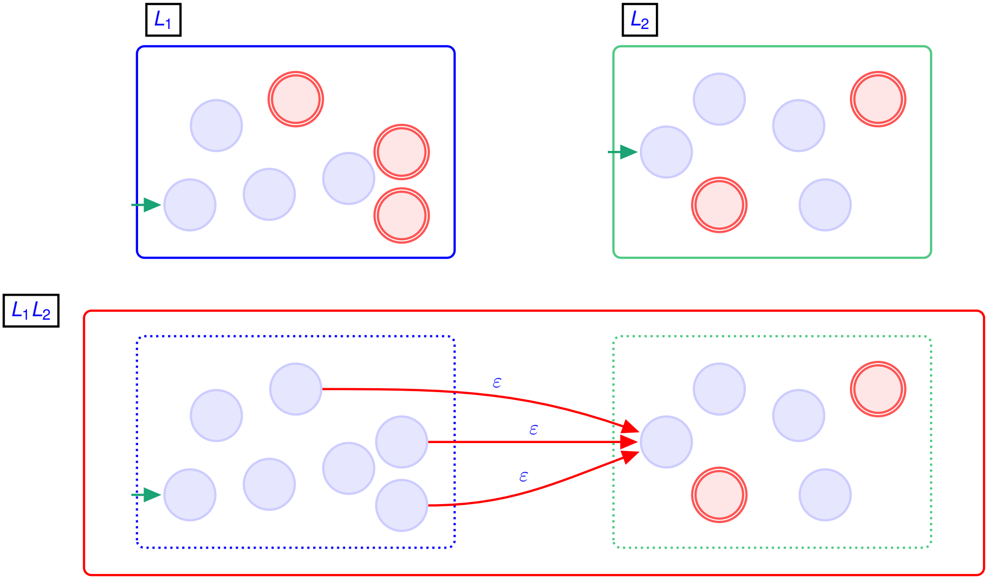
3. $R = \mathtt{a}$ where $a \in \Sigma$ (i.e. $\mathtt{a}$ is a symbol from the alphabet)



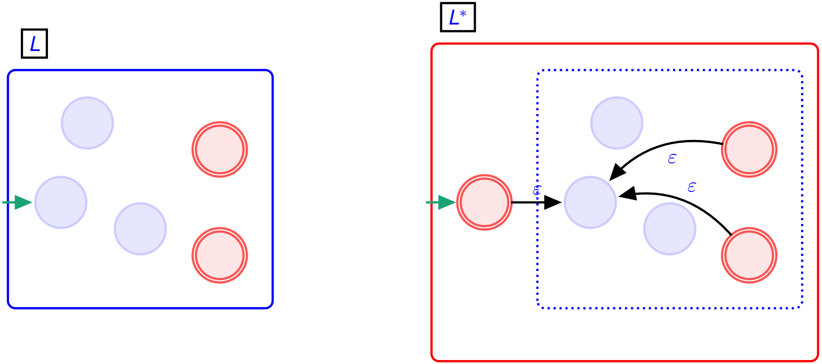
Proof (1/2): RegEx $\rightarrow$ NFA — $A + B$ (Union)



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



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

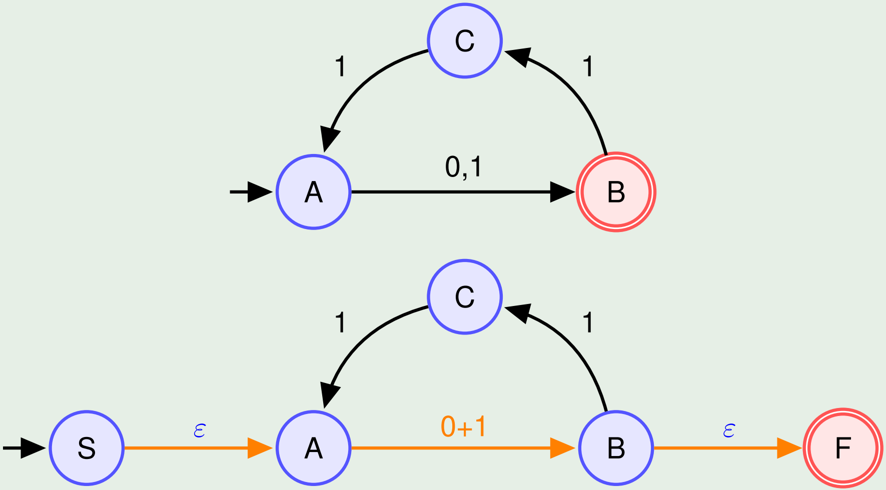
1. Only **one accept state**.
2. The **initial state** has no in-coming transitions.
3. The **accept state** has no out-going transitions.
4. The **transitions** are RegEx's, rather than just symbols from the alphabet.

We can convert a given NFA N into a GNFA in three steps:

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

Proof (2/2): NFA → RegEx —- Converting NFAs into GNFAs

Example (NFA → GNFA)



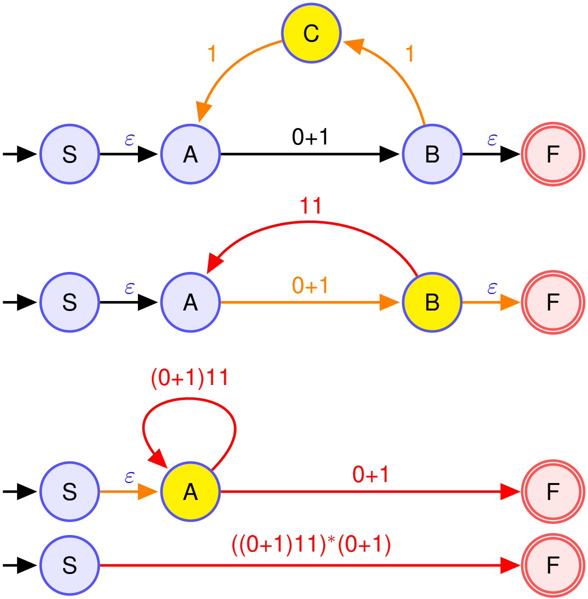
Proof (2/2): NFA $\rightarrow$ RegEx —- Reducing GNFAs into RegEx's

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 equivalent 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



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!)