# COMP2012/G52LAC
# Languages and Computation
# Lecture 6

*Equivalence of Regular Expression and Finite Automata*

Henrik Nilsson

University of Nottingham



## This Lecture (1)

- We have now seen three ways of formally describing potentially infinite languages:
  - Deterministic Finite Automata (DFA)
  - Nondeterministic Finite Automata (NFA)
  - Regular Expressions (RE)
- Because
  - a DFA is a special case of an NFA
  - any NFA can be converted into an equivalent DFA

  DFAs and NFAs describe the same ***class*** of languages: the ***Regular*** languages.



## This Lecture (2)

So, what class of languages do the REs describe? Smaller? Larger? Completely different?

In fact:

- Regular Expressions describe the Regular Languages
- Proof: translation between RE and FA
- This lecture: translation of RE into NFA

Will start by a motivating example.
Time permitting, brief look at another application: scanners. Study details in your own time if of interest.



## Applications (1)

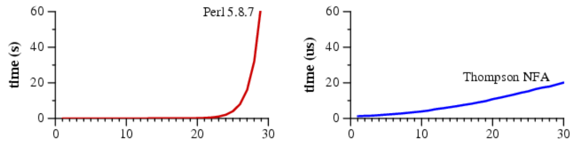
RE to NFA conversion has important practical applications.
The following is a very nice, practically oriented article you should be able to fully appreciate based on what you have learned in G52MAL thus far:

> Russ Cox. *Regular Expression Matching Can Be Simple And Fast (but is slow in Java, Perl, PHP, Python, Ruby, ...)*, January 2007.
> `http://swtch.com/~rsc/regexp/regexp1.html`



## Applications (2)

Underlying message: if you're ignorant about CS theory, your code can perform really poorly.
Example from the paper:




Time to match $(\mathbf{a}+\epsilon)^n\mathbf{a}^n$ against $a^n$

***Note difference of time scale: 60 s vs. 60 $\mu$s!***

`http://en.wikipedia.org/wiki/Thompson's_construction`



## Applications (3)

To quantify:

- Thompson NFA implementation a ***million*** times faster than Perl (5.8.7) when running on a 29-character string.
- Thompson NFA handles a 100-character string in under 200 microseconds; Perl would require over $10^{15}$ years.

How old is the universe?

Current best estimate: ***13.8 billion years*** . . . or about $10^{10}$ years. $10^{15}$ years is a looong time . . .



## Recap: Syntax of Regular Expressions

1. $\emptyset$ is an RE
2. $\epsilon$ is an RE
3. For all $x \in \Sigma$, $\mathbf{x}$ is an RE
   (Handwriting convention: $\underline{x}$ is an RE)
4. If $E$ and $F$ are REs, so is $E + F$
5. If $E$ and $F$ are REs, so is $EF$
6. If $E$ is an REs, so is $E^*$
7. If $E$ is an REs, so is $(E)$

These are ***all*** regular expressions.



## Recap: Semantics of Regular Expr.

1. $L(\emptyset) = \emptyset$
2. $L(\epsilon) = \{\epsilon\}$
3. For all $x \in \Sigma$, $L(\mathbf{x}) = \{x\}$
4. $L(E + F) = L(E) \cup L(F)$
5. $L(EF) = L(E)L(F)$
6. $L(E^*) = L(E)^*$
7. $L(\ (E)\ ) = L(E)$



## Translating RE to NFA (1)

We are going to detail a "Graphical Construction" for converting an RE to an NFA that is suitable for carrying out by hand.

It can be further refined into a fully formal algorithm: see the lecture notes for details.

(Our "Graphical Construction" is a variation of Thompson's Construction. The latter translates into NFA$_\epsilon$: a variation of NFA with a special $\epsilon$-move that does not consume any input. We don't cover NFA$_\epsilon$ in this module.)

## Translating RE to NFA (2)

***Specification:***

Let $N(E)$ denote the NFA that results by applying the graphical construction to an RE $E$. Then the following equation must hold:

$$L(E) \quad = \quad L(N(E))$$

(Note that $L$ is ***overloaded***: the language of an RE to the left, the language of an NFA to the right.)

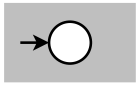
We proceed case by case according to the structure of the syntax of REs.



## RE to NFA, Case $\emptyset$

Recall: $L(\emptyset) = \emptyset$

$N(\emptyset)$:



Note: $L(N(\emptyset)) = \emptyset = L(\emptyset)$; specification satisfied in this case.

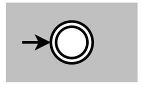
Note: States are given without names for simplicity. Suffice as construction is graphical; states to be named at the end.



## RE to NFA, Case $\epsilon$

Recall: $L(\epsilon) = \{\epsilon\}$

$N(\epsilon)$:



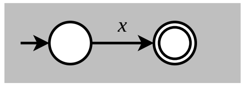
Note: $L(N(\epsilon)) = \{\epsilon\} = L(\epsilon)$; specification satisfied in this case.



## RE to NFA, Case x for $x \in \Sigma$

Recall: For each $x \in \Sigma, L(\mathbf{x}) = \{x\}$

$N(\mathbf{x})$:



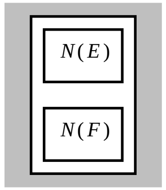

Note: $L(N(\mathbf{x})) = \{x\} = L(\mathbf{x})$; specification satisfied in this case.



## RE to NFA, Case $E + F$ (1)

Recall: $L(E + F) = L(E) \cup L(F)$

$N(E + F)$:




The NFAs $N(E)$ and $N(F)$ in parallel. The initial states of $N(E + F)$ are the union of the initial states of $N(E)$ and $N(F)$.



## RE to NFA, Case $E + F$ (2)

Note: Assuming specification holds for $E$ and $F$,

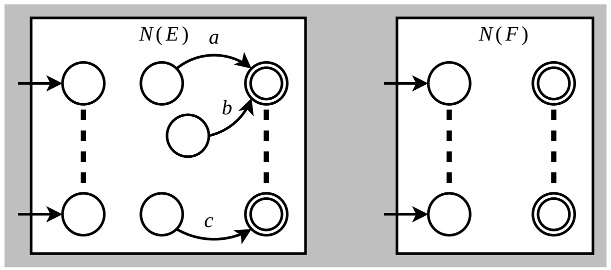
$$\begin{aligned} L(N(E+F)) &= L(N(E)) \cup L(N(F)) \\ &= L(E) \cup L(F) \\ &= L(E+F) \end{aligned}$$

Thus, specification holds in this case. (This is an ***inductive*** case.)



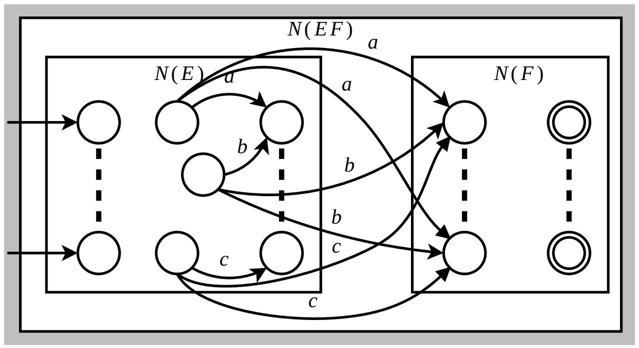
## RE to NFA, Case $EF$ (1)

Sub-case 1: No initial state of $N(E)$ is accepting; i.e. $\epsilon \notin L(N(E))$ (Recall: $L(EF) = L(E)L(F)$)





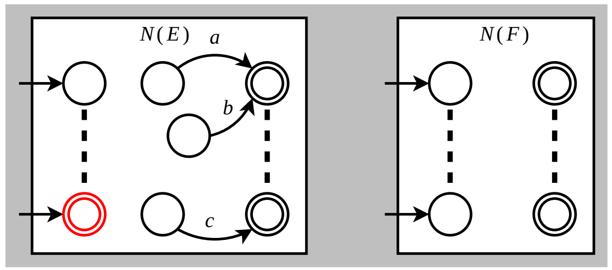

## RE to NFA, Case $EF$ (2)



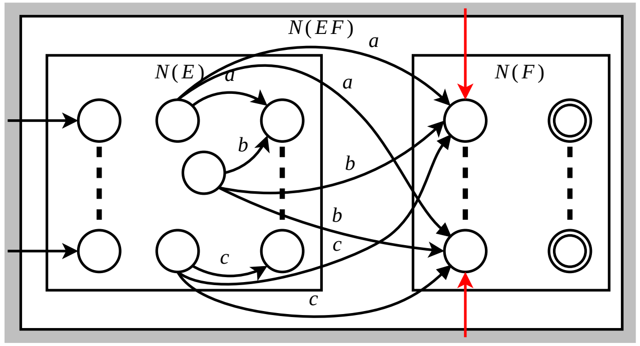



## RE to NFA, Case $EF$ (3)

Sub-case 2: Some initial states of $N(E)$ are accepting; i.e. $\epsilon \in L(N(E))$



## RE to NFA, Case $EF$ (4)






## RE to NFA, Case $EF$ (5)

Note: Assuming specification holds for $E$ and $F$,

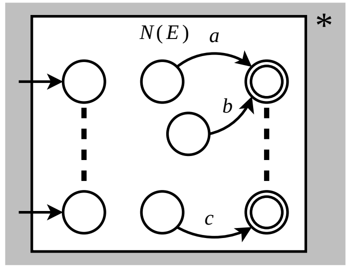
$$\begin{aligned} L(N(EF)) &= L(N(E))L(N(F)) \\ &= L(E)L(F) \\ &= L(EF) \end{aligned}$$

Thus, specification holds in this case.
(This is an ***inductive*** case.)



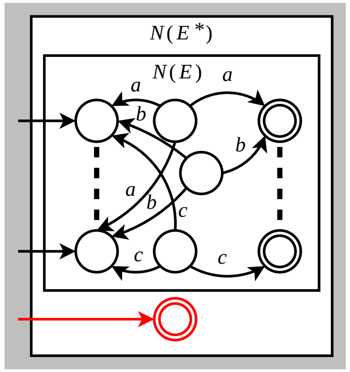
## RE to NFA, Case $E^*$ (1)

(Recall: $L(E^*) = L(E)^*$)






## RE to NFA, Case $E^*$ (2)




Note the additional initial and accepting state that ensures the empty word is accepted.



## RE to NFA, Case $E^*$ (3)

Note: Assuming specification holds for $E$,

$$\begin{aligned} L(N(E^*)) &= L(N(E))^* \\ &= L(E)^* \\ &= L(E^*) \end{aligned}$$

Thus, specification holds in this case.
(This is an ***inductive*** case.)



## RE to NFA, Case $(E)$

(Recall: $L(\ (E)\ ) = L(E)$)

$N(\ (E)\ ) = N(E)$

Note: Assuming specification holds for $E$,

$$\begin{aligned} L(N(\ (E)\ )) &= L(N(E)) \\ &= L(E) \\ &= L(\ (E)\ ) \end{aligned}$$

Thus, specification holds in this case.
(This is an ***inductive*** case.)



## Example

Systematically construct an NFA for the regular expression:

$$(\mathbf{a}+\mathbf{b})^*\mathbf{c}$$

("zero or more $a$s or $b$s, followed by a single $c$")

Use the "graphical construction". On the white board.



## Scanning (1)

- The first stage of many real-world language processing tasks, such as a compiler, is to group individual characters into language-specific symbols called ***Lexemes*** or ***Tokens***:
  - Keywords (like `if`, `then`, `while`)
  - Literals (like `42`, `3.14`, `'A'`, `"abc"`)
  - Special symbols and separators (like `:=`, `(`, `;`)
  - . . .
- This process is called ***Lexical Analysis*** or ***Scanning***, and is performed by a ***Scanner***.



## Scanning (2)

- Commonly, ***white space*** and ***comments*** are understood as ***token separators***.
- An additional task of the scanner is often to ***discard*** white space and comments as they usually serve no purpose after the scanning.
- Regular expressions is the most commonly used formalism for describing the ***Lexical Syntax*** of a language; i.e. the syntax of the tokes, white space, and comments.
- In essence, a scanner is thus a ***finite automaton***.

# Scanning (3)

- There are many famous so called ***scanner generators***; e.g. Lex, Flex: given regular expressions describing the lexical syntax, they produce a scanner for the language.
- Internally, they use Thompson's construction (or similar).