## From regular expressions to $\epsilon$-NFAs

We let behind the regular expressions when we introduced informally the transition diagrams for the token recognition.

Let us show now that regular expressions, used in lexers to specify tokens, can be converted to $\epsilon$-NFAs, so to DFA. This proves that *regular languages are recognisable languages*.

Actually, it is possible to prove that any $\epsilon$-NFA can be converted to a regular expression denoting the same language, but we will not do so.

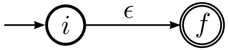
Therefore, keep in mind that **regular languages are recognisable languages**. In other words, using a regular expression or a finite automaton is only a matter of convenience.

## From regular expressions to $\epsilon$-NFAs (cont)

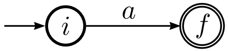
The construction we present here to build an $\epsilon$-NFA from a regular expression is called **Thompson's construction**.

Let us first associate an $\epsilon$-NFA to the basic regular expressions.

- For the expression $\epsilon$, construct the following NFA, where *i* and *f* are **new** states



- For $a \in \Sigma$, construct the following NFA, where *i* and *f* are **new** states

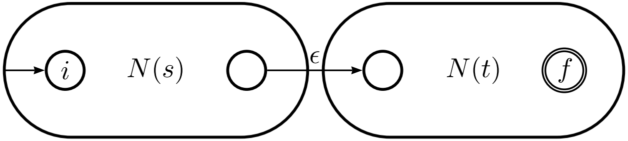


## From regular expressions to $\epsilon$-NFAs (cont)

Now let us associate NFAs to complex regular expressions.

Assume $N(s)$ and $N(t)$ are the NFAs for regular expressions $s$ and $t$.

- For the regular expression $st$, construct the following NFA $N(st)$, where **no new state** is created:

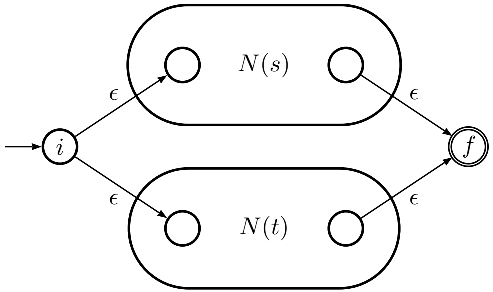


The final state of $N(s)$ becomes a normal state, as well as the initial state of $N(t)$.

This way only remains a unique initial state $i$ and a unique final state $f$.

## From regular expressions to $\epsilon$-NFAs (cont)

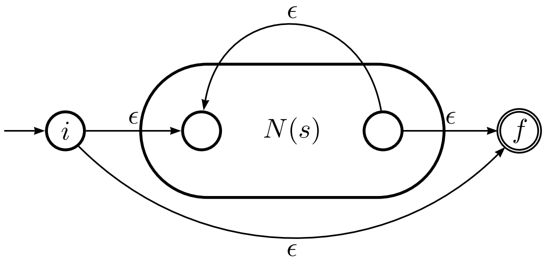
- For the regular expression $s \mid t$, construct the following NFA $N(s \mid t)$



where $i$ and $f$ are **new** states. Initial and final states of $N(s)$ and $N(t)$ become normal.

## From regular expressions to $\epsilon$-NFAs (cont)

- For the regular expression $s^\star$, construct the following NFA $N(s^\star)$, where $i$ and $f$ are **new** states:



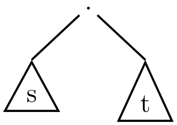
Note that we added two $\epsilon$ transitions and that the initial and final states of $N(s)$ become normal states.

## From regular expressions to $\epsilon$-NFAs (cont)

But how do we apply these simple rules when we have a complex regular expression, having many level of nested parentheses etc?

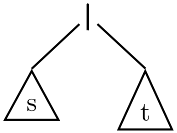
Actually, the **abstract syntax tree** of the regular expression direct, i.e., orders, the application of the rules.

If the syntax tree has the shape



then we construct first $N(s)$, $N(t)$ and finally $N(st)$.

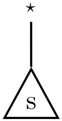
If the syntax tree has the shape



then we construct first $N(s)$, $N(t)$ and finally $N(s \,|\, t)$.

## From regular expressions to $\epsilon$-NFAs (cont)

If the syntax tree has the shape

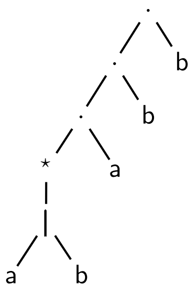


then we construct first $N(s)$ and finally $N(s^\star)$.

This pattern-matchings are applied first at the **root** of the abstract syntax tree of the regular expression.

## From regular expressions to $\epsilon$-NFAs/Exercise

Consider the regular expression $(\mathrm{a} \mid \mathrm{b})^{\star}\mathrm{abb}$ and its abstract syntax tree



Apply the previous rules to build the corresponding $\epsilon$-NFA.