Simulation of an NFA by a DFA

Let $N = (Q_N, \Sigma, \delta_N, q_0^N, F_N)$ be a NFA. The equivalent DFA D is obtained from the so-called **powerset construction** (also called “subset construction”.) We define

$$D = (Q_D, \Sigma, \delta_D, q_0^D, F_D),$$

where. . .

Simulation of an NFA by a DFA

- The alphabet of D is that of N.
- The states of D are sets of states of N:

$$Q_D = P(Q_N)$$

- The initial state q_0^D of D is $\{q_0^N\}$.

Simulation of an NFA by a DFA

- The final states of D are those sets that contain the final state of N:

$$F_D = \{S \in P(Q_N) \mid S \cap F_N \neq \emptyset\}$$

- The transition function of D arises from the transition function of N as follows:

$$\delta_D(S, a) = \bigcup_{q \in S} \delta_N(q, a)$$

That is, $\delta_D(S, a)$ is the set of all states of N that are reachable from some state q via a.

Proposition about the simulation

Proposition. For every NFA N, there is a DFA D such that $L(D) = L(N)$.

The proof is in last week's handout.

Regular expressions

Regular expressions: motivation

- Useful for describing text patterns (with wildcards etc.).
- Used e.g. for text search in the text editor “Emacs” and in the Unix search command “grep”.
- Used in compilers for recognizing **tokens** of programming languages, e.g. identifiers, floating-point-numbers, and so on. (See compilers lecture.)

First example

The regular expression

$$01^* + 10^*$$

denotes the language consisting of all strings that are either a single 0 followed by any number of 1's, or a single 1 followed by any number of 0's.

Operations on languages

Before describing the regular-expression notation, we need to define the operations on languages that the operators of regular expressions represent.

Concatenation

- The **concatenation** $L \cdot L'$ (or just LL') of languages L and L' is defined to be the set of strings ww' where $w \in L$ and $w \in L'$.
- For example, if $L = \{001, 10, 111\}$ and $L' = \{\varepsilon, 001\}$, then $LL' = \{001, 10, 111, 001001, 10001, 111001\}$.

Self-concatenation

- For a language L, we write L^n for

$$\underbrace{L \cdot L \cdot \cdots \cdot L}_{n \text{ times}}$$

- That is, L^n is the language that consists of strings $w_1 w_2 \dots w_n$, where each w_i is in L.
- For example, if $L = \{\varepsilon, 001\}$, then $L^3 = \{\varepsilon, 001, 001001, 001001001\}$.
- Note that $L^1 = L$. The language L^0 is defined to be $\{\varepsilon\}$.

Closure (Kleene-star)

- The **closure** (or **star** or **Kleene closure**) L^* of a language L is defined to be

$$L^* = \bigcup_{n \geq 0} L^n$$

- That is, L^* is the language that consists of strings $w_1 w_2 \dots w_k$, where k is **any** non-negative integer and each w_i is in L.
- E.g. if $L = \{0, 11\}$, then L^* consists of all strings such that the 1's come in pairs, e.g. 011, 11110, and ε, but not 01011 or 101.

Regular expressions: definition

Definition. The **regular expressions** over an alphabet Σ are defines as follows:

- Every symbol $a \in \Sigma$ is a regular expression.
- If E and E' are regular expressions, then so is $E + E'$ and $E \cdot E'$. (We shall abbreviate the latter by EE'.)
- If E is a regular expressions, then so is E^*.
- The symbol ε is a regular expression.
- The symbol $\emptyset$ is a regular expression.

Semantics of regular expressions

(Remark: “semantics” is the technical term for “meaning”.)

Regular expression E	denoted language $L(E)$
$a \in \Sigma$	$\{a\}$
$E + E'$	$L(E) \cup L(E')$
$E \cdot E'$	$L(E) \cdot L(E')$
E^*	$(L(E))^*$
ε	$\{\varepsilon\}$
$\emptyset$	the empty language, $\emptyset$

Example

- Suppose you want to search some messy text file for the street parts of addresses, e.g. “Milsom Street” or “Wells Road”.
- Let $[A - Z]$ stand for $A + B + \cdots + Z$.
- Let $[a - z]$ stand for $a + b + \cdots + z$.
- You may want to use a regular expression like

$$[A{-}Z][a{-}z]^* \;\; (Street{+}St.{+}Road{+}Rd.{+}Lane)$$

- Expressions like this are accepted e.g. by the UNIX command `grep`, the EMACS text editor, and various other tools.

Exercises

Write regular expressions for the following languages:

1. The set of strings over alphabet $\{a, b, c\}$ with at least one a and at least one b.
2. The set of strings of 0's and 1's whose tenth symbol from the right end is 1.
3. The set of strings of 0's and 1's with at most on pair of consecutive 1's.

Exercises

Write regular expressions for the following languages:

1. The set of all strings of 0's and 1's such that every pair of adjacent 0's appears before any pair of adjacent 1's.
2. The set of strings of 0's and 1's whose number of 0's is divisible by five.

Exercise

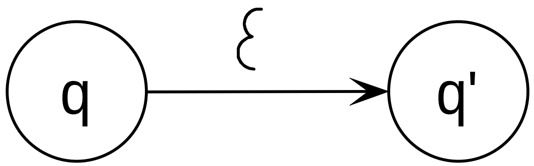
For any alphabet Σ, which are the subsets S of Σ^* such that the set S^* is finite?

Regular expressions and FA's: overview

- As we shall see, for every regular expression E, there is an NFA (and therefore also a DFA) that accepts the language defined by E.
- Tools that scan text for regular expressions work in this way.
- Also, for every DFA (and therefore for every NFA) A, there is a regular expression that denotes the language accepted by A.
- So finite automata and regular expressions are equivalent with respect to the definable languages.

NFA's with ε-transitions

- For simulating regular expressions, it is helpful to introduce **NFA's with ε-transitions**, or ε**-NFA's in short**.
- The only difference between ε-NFA's and NFA's is that the former can make spontaneous transitions, i.e. transitions that use up no input—technically speaking, the empty string ε.



NFA's with ε-transitions

- More formally, an ε-NFA differs from an NFA only in that its transition function also accepts ε as an argument:

$$\delta : Q \times (\Sigma \cup \varepsilon) \to P(Q)$$

- It can be shown by some **modified powerset construction** that for every ε-NFA there is a DFA accepting the same language (see Hopcroft/Motwani/Ullman).

From regular expressions to FA's

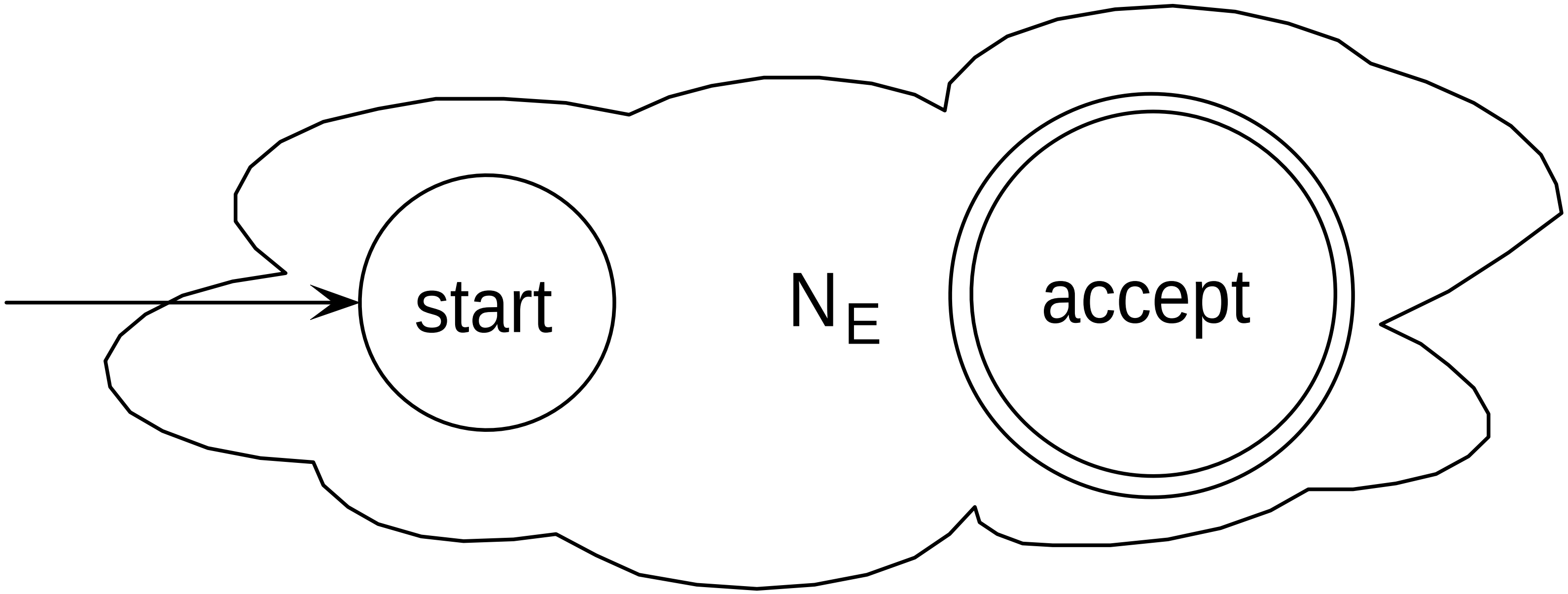
Formally, we shall prove:

Proposition. For every regular expression E, there is an ε-NFA N_E such that $L(N_E) = L(E)$.

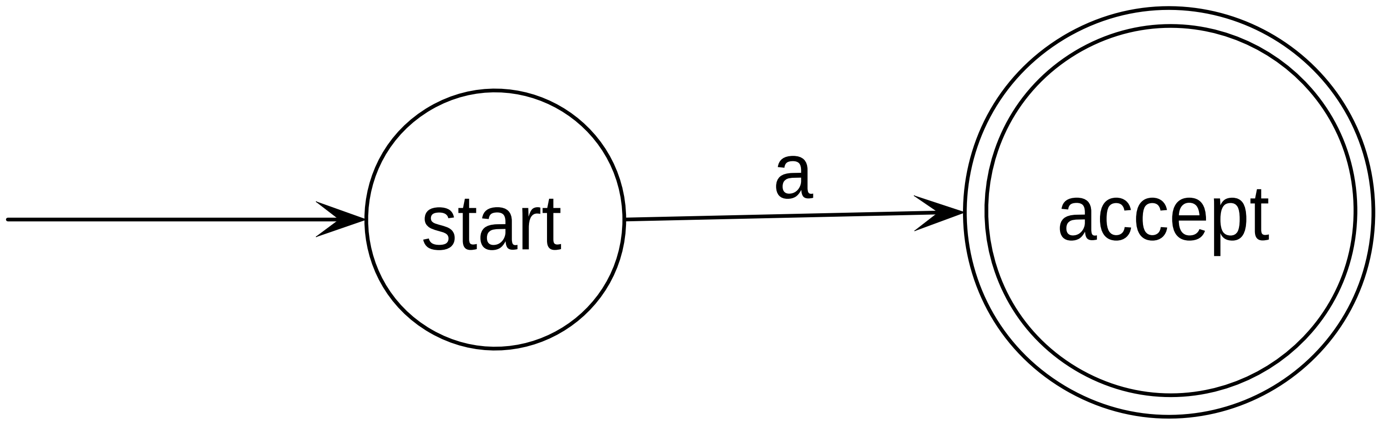
We shall see how this works on the next few slides.

The ε-NFA N_E of a regular expression E

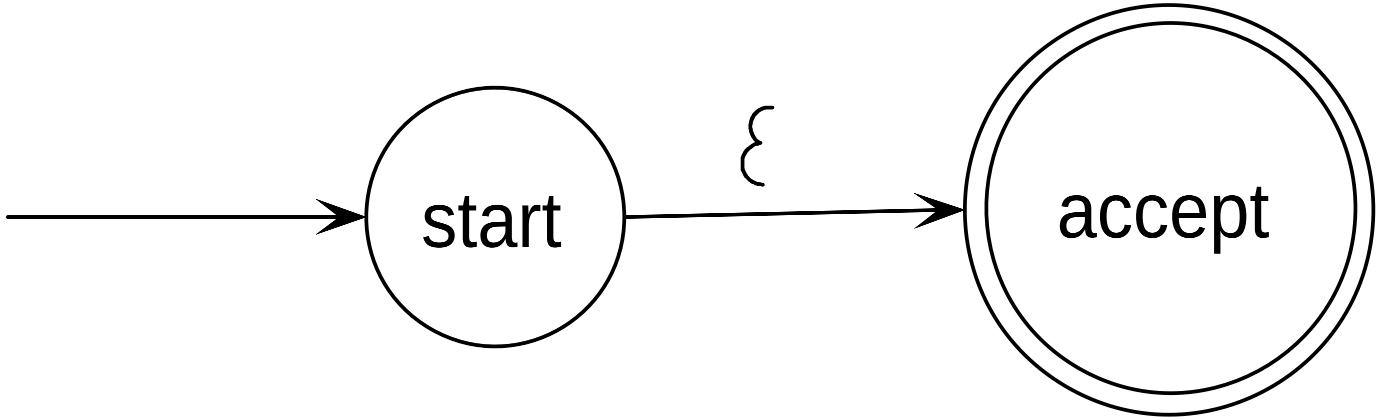
For every regular expression E, we shall build an ε-NFA N_E with exactly one accepting state, from which no further transitions are possible:



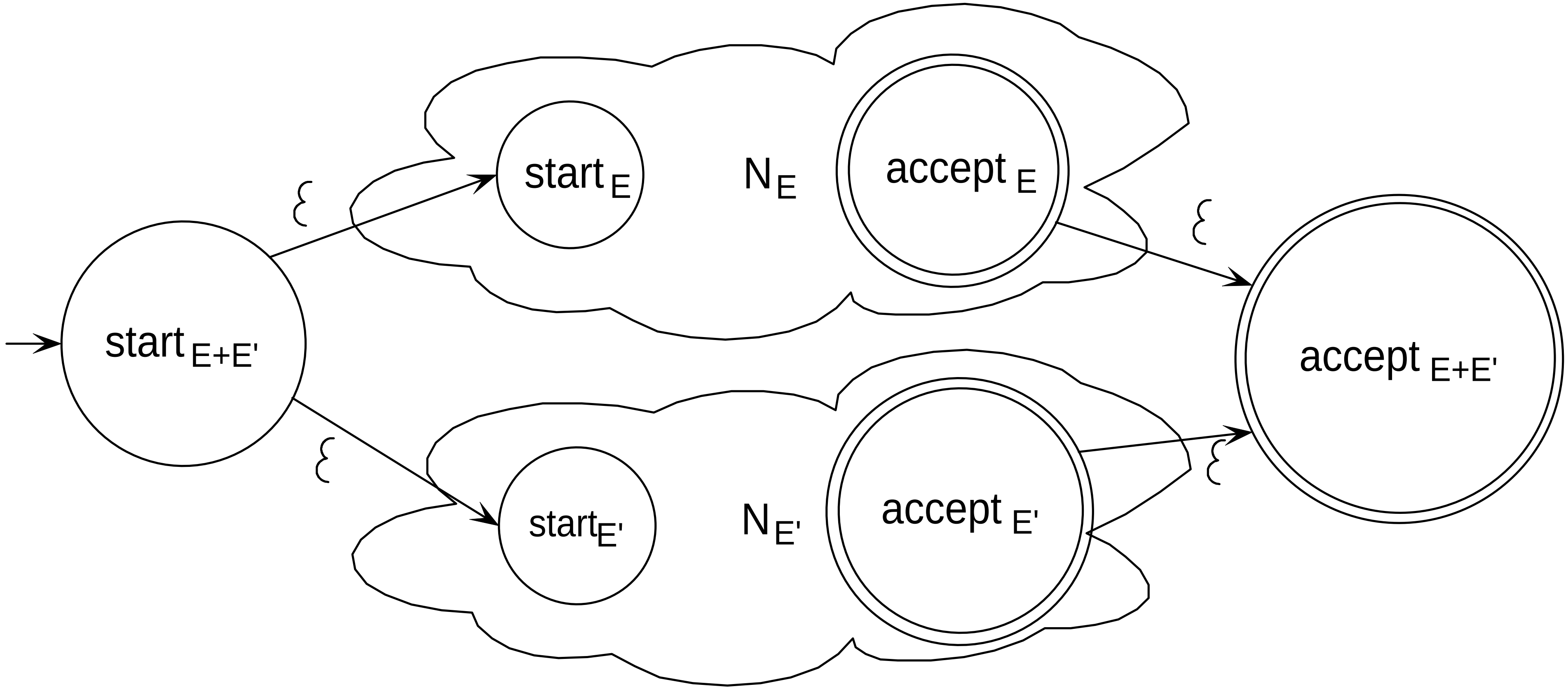
N_E **for** $E = a \in \Sigma$



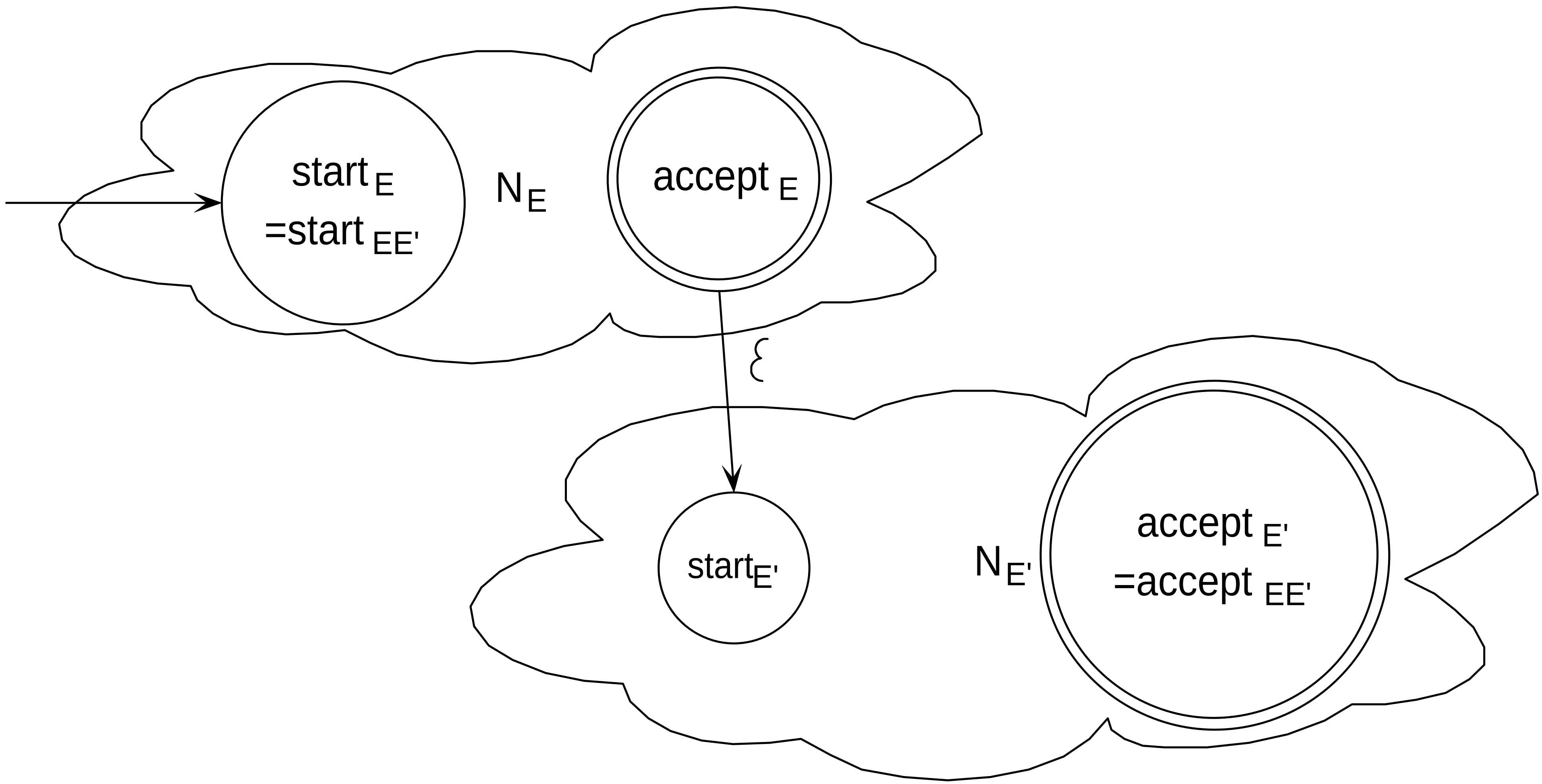
N_E for $E = \varepsilon$



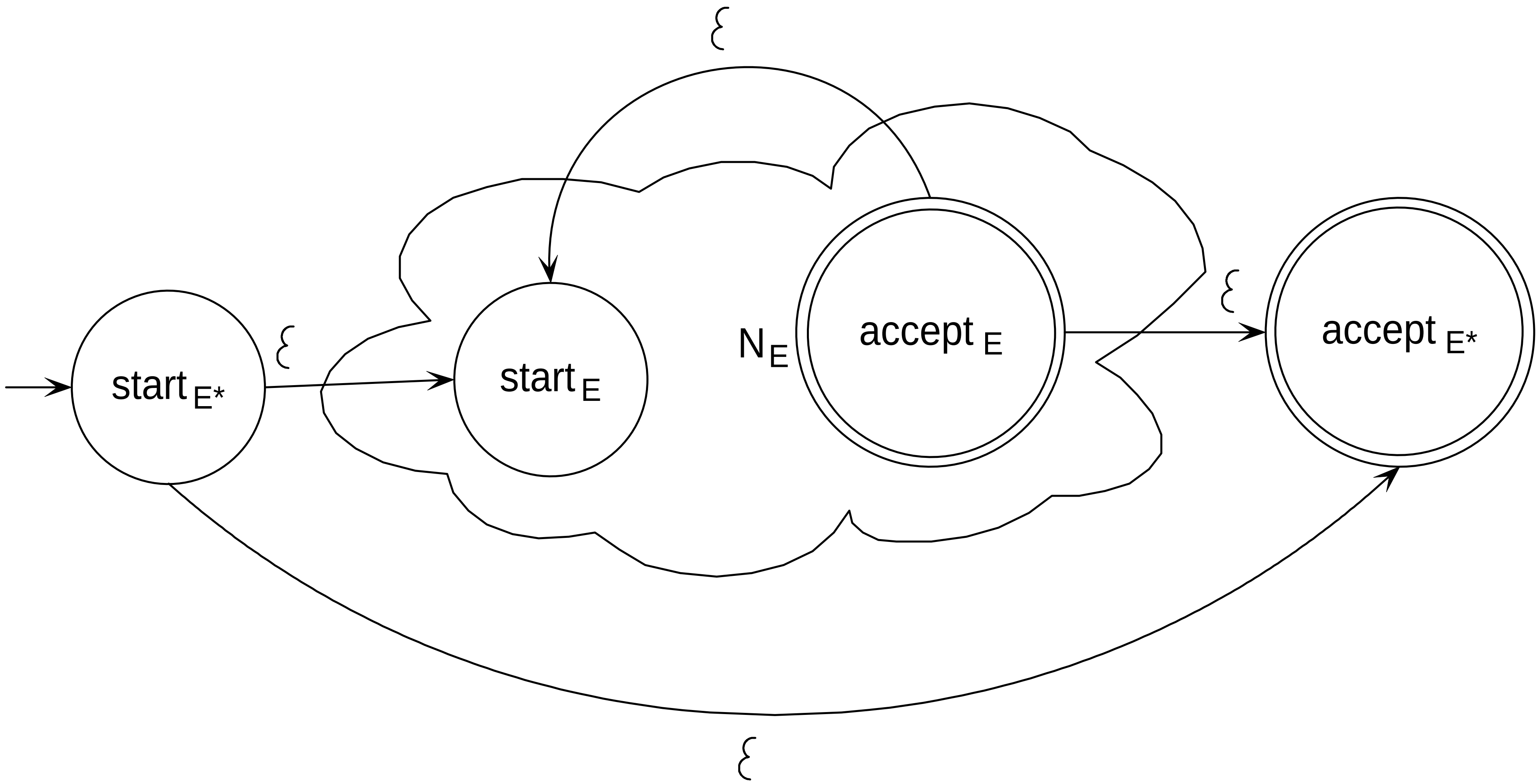
$N_{E+E'}$



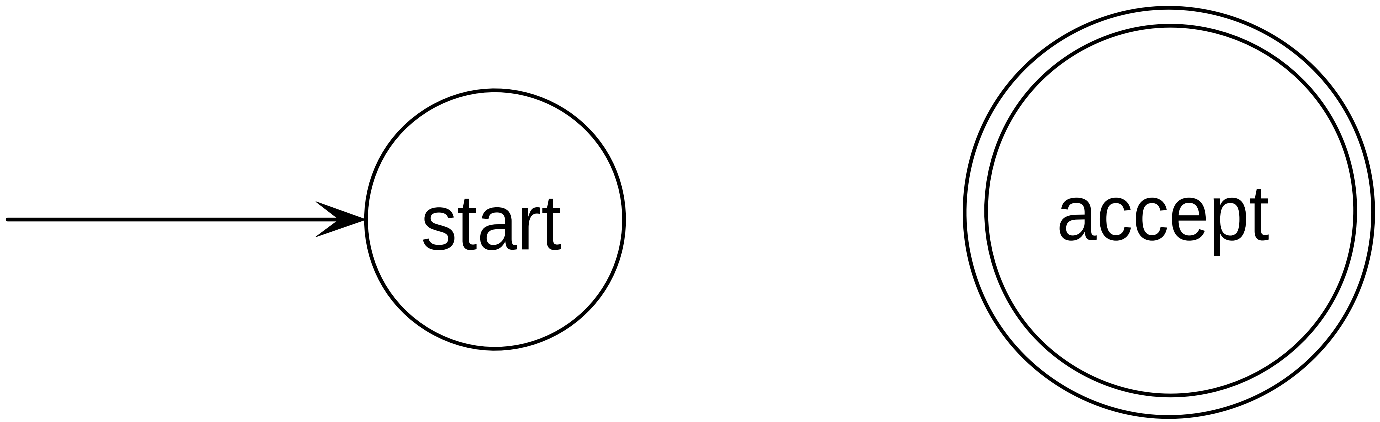
$N_{E \cdot E'}$



N_{E*}



N_E for $E = \emptyset$



There is no way to get from the start state to the accepting state.