Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Talen en Compilers

2019 - 2020, period 2

Jurriaan Hage

Department of Information and Computing Sciences
Utrecht University

2019-12-10

10. Finite state automata

10.1 The Chomsky hierarchy

Context-free languages

The languages we dealt with until now were mostly **context-free** languages:

- can be described using a context-free grammar,
- can be parsed relatively easily (for instance, using parser combinators),
- resulting parsers need polynomial time and space (worst case in between quadratic and cubic time, often not much worse than linear).

The rest of the course: classes of languages and/or grammars that allow more efficient parsing.

The Chomsky hierarchy

- Type 0: **unrestricted** grammars, **recursively enumerable** languages
  - Require a Turing machine for acceptance
  - As expressive as any other computational formalism
- Type 1: **context-sensitive** grammars and languages
- Type 2: **context-free** grammars and languages
  - Parsed using a pushdown automata in polynomial time
- Type 3: **regular** grammars and languages
  - Recognized by a finite state automata
  - Require only linear time and constant space
  - Equivalent to regular expressions

Questions

For **regular** and **context-free** languages:

- What are the restrictions on that level of the hierarchy?
  - **Closure**: what happens if we combine two languages into a single one via union, sequence, or any other operation?
  - **Pumping lemmas**: how to detect that a language does **not** belong to a certain level.
- How do we parse such languages efficiently?
  - Relation to **automata**-based formalisms.
  - For context-free language, further restrictions: LL and LR.

A note on English

Finite state automaton = eindigetoestandsautomaat

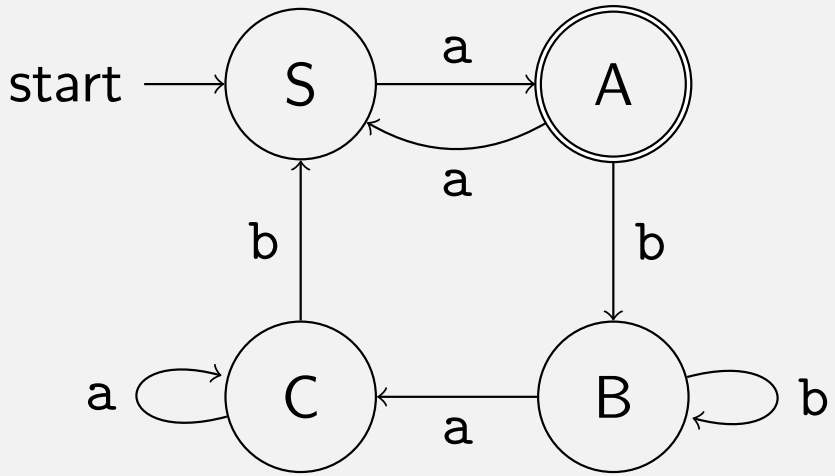
But...

One automat**on** = één automaat (= un autómata)

Two automat**a** = twee automaten (= dos autómatas)

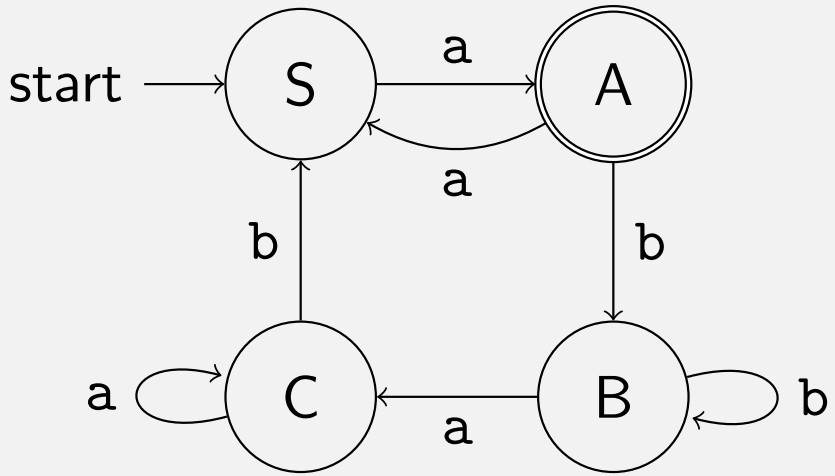
10.2 Deterministic finite state automata (DFA)

Deterministic finite state automata (DFA)

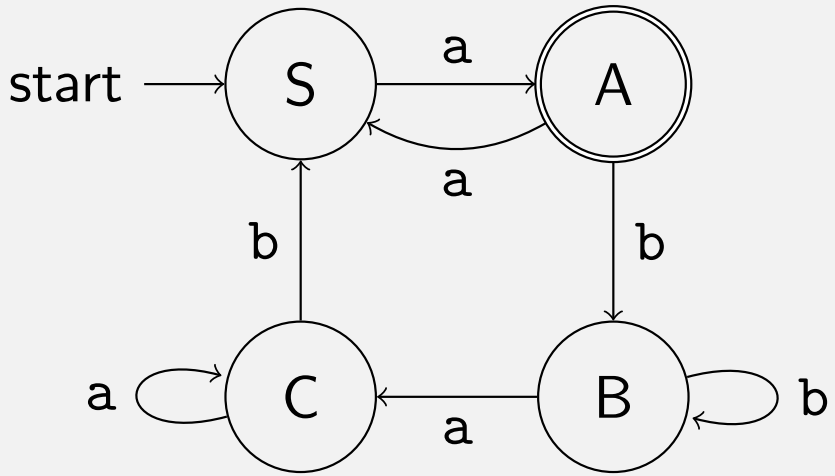


Deterministic finite state automata (DFA)

- Input alphabet:
  $X = \{a, b\}$

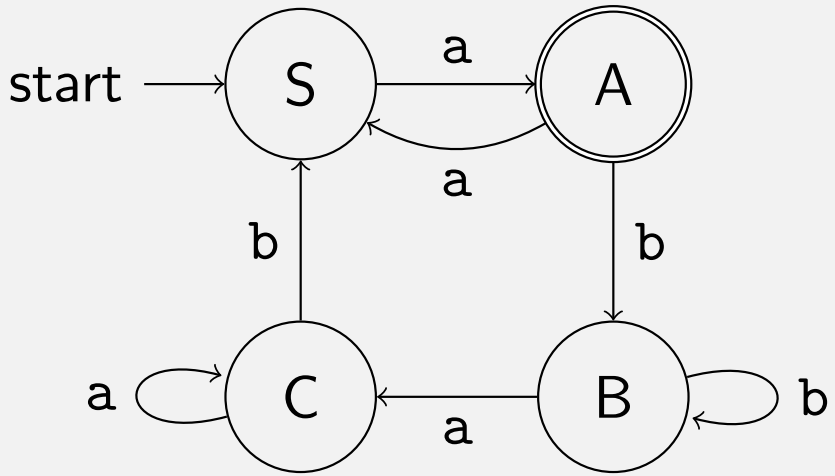


Deterministic finite state automata (DFA)



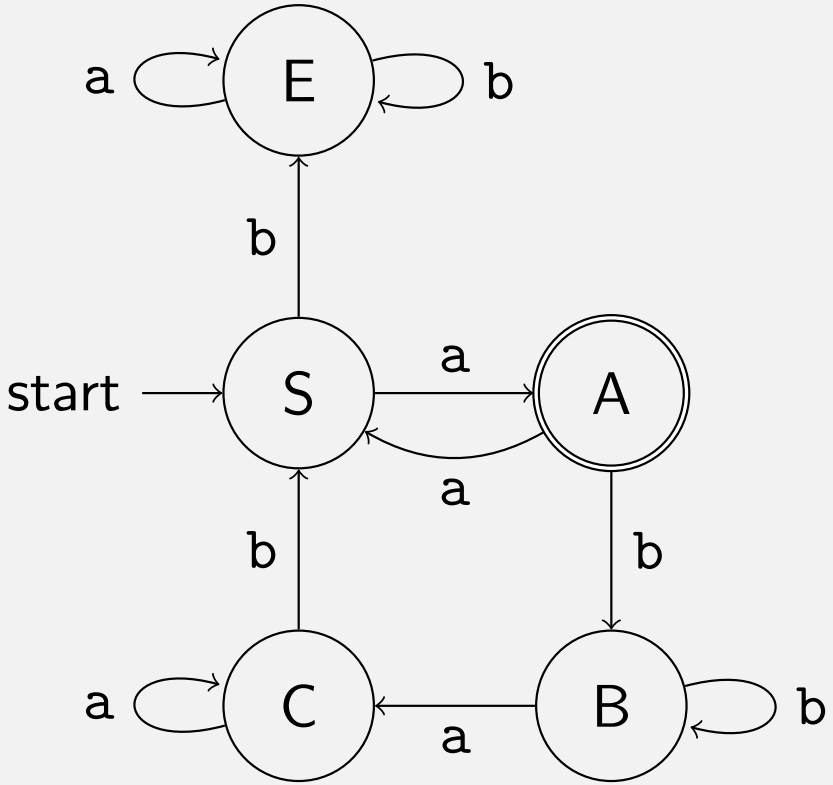
- Input alphabet:
  $X = \{a, b\}$
- States:
  $Q = \{S, A, B, C\}$

Deterministic finite state automata (DFA)



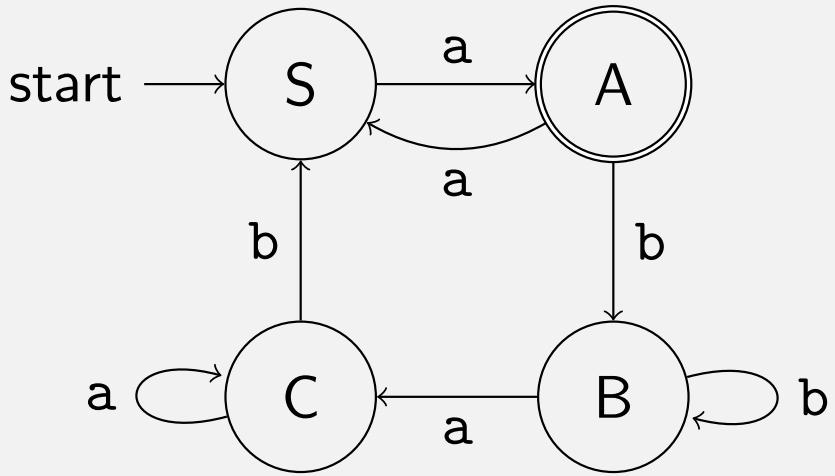
- Input alphabet:
  $X = \{a, b\}$
- States:
  $Q = \{S, A, B, C\}$
- Transitions:
  $d :: Q \to X \to Q$

Deterministic finite state automata (DFA)



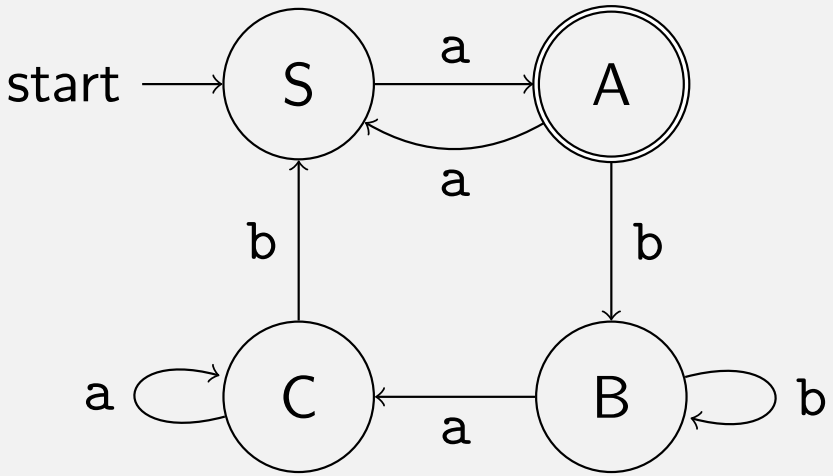
- Input alphabet:
  $X = \{a, b\}$
- States:
  $Q = \{S, A, B, C\}$
- Transitions:
  $d :: Q \to X \to Q$

Deterministic finite state automata (DFA)



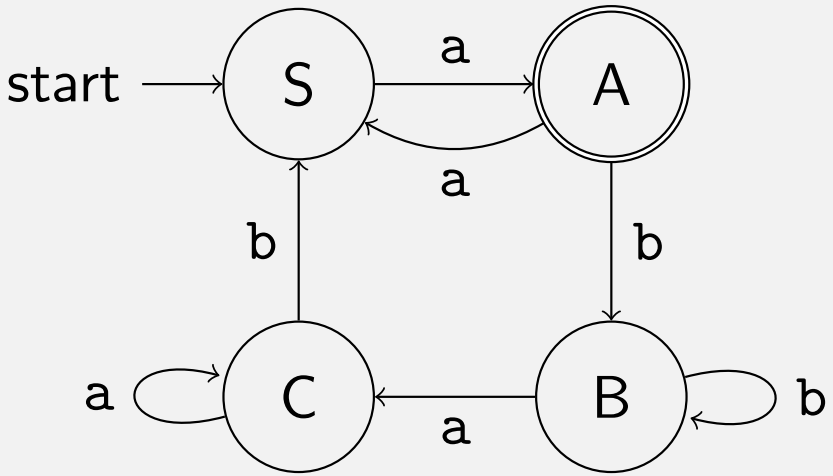
- Input alphabet:
  $X = \{a, b\}$
- States:
  $Q = \{S, A, B, C\}$
- Transitions:
  $d :: Q \to X \to Q$

Deterministic finite state automata (DFA)



- Input alphabet:
  $X = \{a, b\}$
- States:
  $Q = \{S, A, B, C\}$
- Transitions:
  $d :: Q \to X \to Q$
- Start state:
  S (where $S \in Q$)

Deterministic finite state automata (DFA)



- Input alphabet:
  $X = \{a, b\}$
- States:
  $Q = \{S, A, B, C\}$
- Transitions:
  $d :: Q \to X \to Q$
- Start state:
  S (where $S \in Q$)
- Accepting states:
  $F = \{A\}$
  (where $F \subseteq Q$)

Definition of a DFA

A DFA is given by a tuple (X, Q, d, S, F):

- an input alphabet X (sometimes we use Σ),
- a set of states Q,
- a transition function d (or δ) of type $Q \to X \to Q$,
- a start (or initial) state $S \in Q$,
- a set of accepting (or final) states $F \subseteq Q$.

Definition of a DFA

A DFA is given by a tuple (X, Q, d, S, F):

- an input alphabet X (sometimes we use Σ),
- a set of states Q,
- a transition function d (or δ) of type $Q \to X \to Q$,
- a start (or initial) state $S \in Q$,
- a set of accepting (or final) states $F \subseteq Q$.

Sometimes, when X and Q are clear from the context, the triple (d, S, F) is sufficient to specify a DFA.

Running a DFA

Essentially, traverse the arrows in the DFA:

```
dfa :: (Q → X → Q) → Q → [X] → Q
dfa d q []       = q
dfa d q (x : xs) = dfa d (d q x) xs
```

Question

Does this function look familiar?

Running a DFA

Essentially, traverse the arrows in the DFA:

```
dfa :: (Q → X → Q) → Q → [X] → Q
dfa d q []       = q
dfa d q (x : xs) = dfa d (d q x) xs
```

Question

Does this function look familiar?

```
dfa = foldl
```

Running-time guarantees of a DFA

Running a DFA is possible in:

- **linear** time (w.r.t. the length of the word), and
- **constant** space (which state you are in).

Language accepted by a DFA

A word xs is **accepted** or **recognized** by a DFA if running the DFA on the word, starting in the start state S, yields an accepting state.

```
dfaAccept :: [X] → (Q → X → Q, Q, Set Q) → Bool
dfaAccept xs (d, s, fs) = dfa d s xs `member` fs
```

All words that are accepted by the DFA $A = (d, S, F)$ are called the **language** of such a DFA.

$$L\ (A) = \{w \in [X] \mid \text{dfaAccept w A}\}$$

Important! The same language may be accepted by several automata.

Exercise

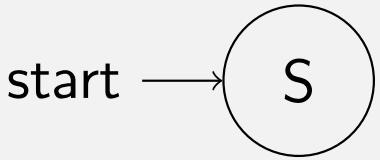
Question

Can the empty language be described by a DFA?

Exercise

Question

Can the empty language be described by a DFA?



An automaton without accepting states is possible.

Exercise

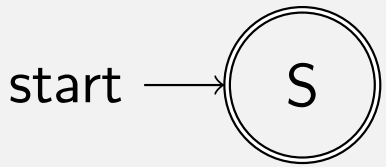
Question

Can the language $\{\varepsilon\}$ be described by a DFA?

Exercise

Question

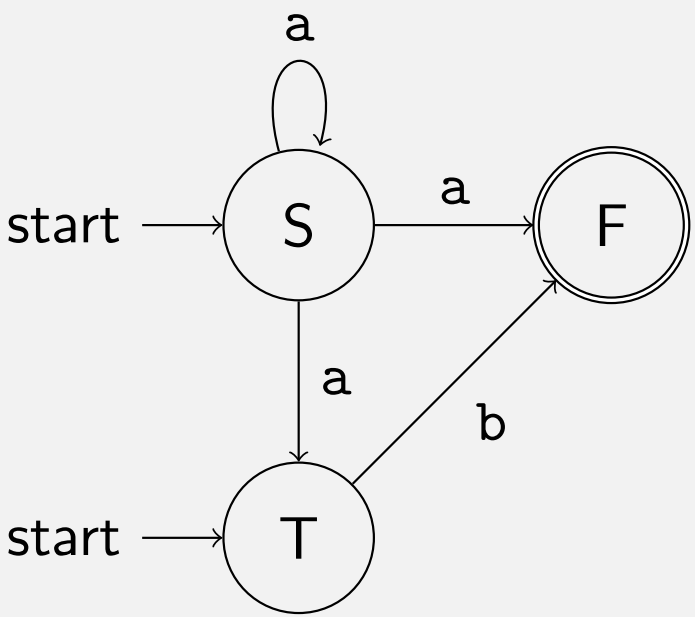
Can the language $\{\varepsilon\}$ be described by a DFA?



In general, any automaton where the starting state is accepting will accept the empty word (and possibly other words).

10.3 Non-deterministic finite state automata (NFA)

NFA example



Nondeterministic finite state automata (NFA)

Similar to DFA, but:

- Potentially multiple start states.
- Potentially multiple transitions for the same terminal from a given state.

Nondeterministic finite state automata (NFA)

Similar to DFA, but:

- Potentially multiple start states.
- Potentially multiple transitions for the same terminal from a given state.

Formally a tuple (X, Q, d, S, F):

- an input alphabet X,
- a set of states Q,
- a transition function d of type $Q \to X \to \text{Set } Q$,
- a **set of** start state**s** $S \subseteq Q$,
- a set of accepting states $F \subseteq Q$.

Interpretation using choices

- We choose any start state.
- When processing a terminal, we choose one of the possible transitions for that terminal at that state and thereby end up with a new state.
- A word is accepted if there is a sequence of choices that gets us to an accepting state.

Interpretation using a set of all possible choices

- At any time, a set of states in the NFA is active. We start with the set of start states.
- When we process a terminal, we take all possible actions from all current states and thereby end up with a new set of states.
- A word is accepted if the set of states that is active after processing the word contains at least one accepting state.

Running an NFA (monadic)

If Set was a monad, we could write:

```
nfa :: (Q → X → Set Q) → Set Q → [X] → Set Q
nfa d qs []       = qs
nfa d qs (x : xs) = do q ← qs          -- For every state
                       nexts ← d q x -- Compute next states
                       nfa d nexts xs -- And go on
```

Running an NFA

```
nfa :: (Q → X → Set Q) → Set Q → [X] → Set Q
nfa d qs []       = qs
nfa d qs (x : xs) = nfa d (join (map (flip d x) qs)) xs
```

where join is the concat for sets and computes the union of a set of sets:

```
join :: Set (Set Q) → Set Q
```

Language accepted by an NFA

We get to at least one accepting state:

$$\begin{array}{l}\mathsf{nfaAccept} :: [\mathsf{X}] \to (\mathsf{Q} \to \mathsf{X} \to \mathsf{Set\ Q}, \mathsf{Set\ Q}, \mathsf{Set\ Q}) \to \mathsf{Bool} \\ \mathsf{nfaAccept\ xs\ (d, ss, fs)} = \mathsf{not\ (null\ (nfa\ d\ ss\ xs\ `intersect`\ fs))}\end{array}$$

$$\begin{array}{rcl}\mathsf{nfaAccept\ xs\ (d,\ ss,\ fs)} & = & \exists_{\mathsf{f} \in \mathsf{fs}}\ \mathsf{f} \in \mathsf{nfa\ d\ ss\ xs} \\ & = & \exists_{\mathsf{f} \in \mathsf{nfa\ d\ ss\ xs}}\ \mathsf{f} \in \mathsf{fs} \\ & = & (\mathsf{fs} \cap \mathsf{nfa\ d\ ss\ xs} \neq \emptyset)\end{array}$$

The language accepted by a NFA A is the set of all words accepted by it:

$$\mathsf{L\ (A)} = \{\mathsf{w} \in [\mathsf{X}] \mid \mathsf{nfaAccept\ w\ A}\}$$

10.4 NFAs vs. DFAs

Claim

DFAs and NFAs accept the same languages.

Claim

DFAs and NFAs accept the same languages.

- from a DFA we can build an NFA which accepts the same language (the easy part),
- from an NFA we can build a DFA which accepts the same language (the hard part)

From DFA to NFA

Every DFA (d, S, F) is trivially an NFA.

From DFA to NFA

Every DFA (d, S, F) is trivially an NFA.

- The language and the set of states are kept the same.
- The start state S becomes the one-element set of start states $\{S\}$.
- The transition function is changed such that it returns singleton sets:

```
d' :: Q → X → Set Q
d' q x = singleton (d q x)
```

It is easy to show that the resulting NFA accepts the same language.

From NFA to DFA (1)

Every NFA (X, Q, d, S, F) can be made deterministic.

From NFA to DFA (1)

Every NFA (X, Q, d, S, F) can be made deterministic.

The construction is called the **powerset construction**:

- The language X is kept the same.
- The new states Q' correspond to **subsets** of states in Q:

$$Q' = \text{subsets}(Q) = \mathcal{P}(Q) = 2^Q$$

- The new start state corresponds to the original set.
- A state is accepting if it contains at least one original accepting state:

$$F' = \{q \in Q' \mid q \cap F \neq \emptyset\}$$

From NFA to DFA (2)

Every NFA (X, Q, d, S, F) can be made deterministic.

The construction is called the **powerset construction**:

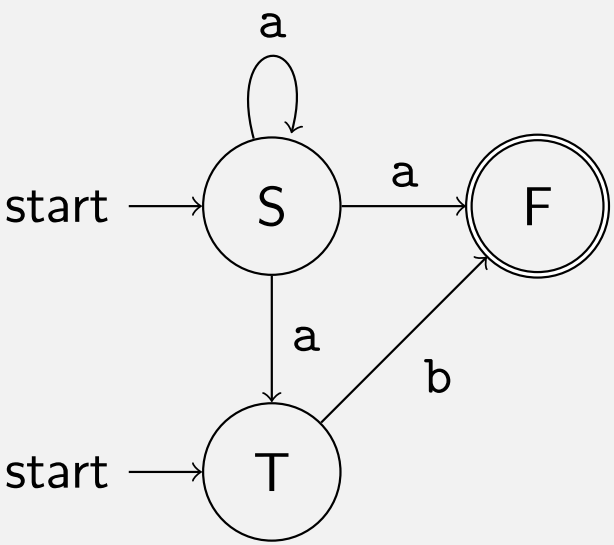
- ...
- The new transition function arises from applying the transition function over all the states in the subset:

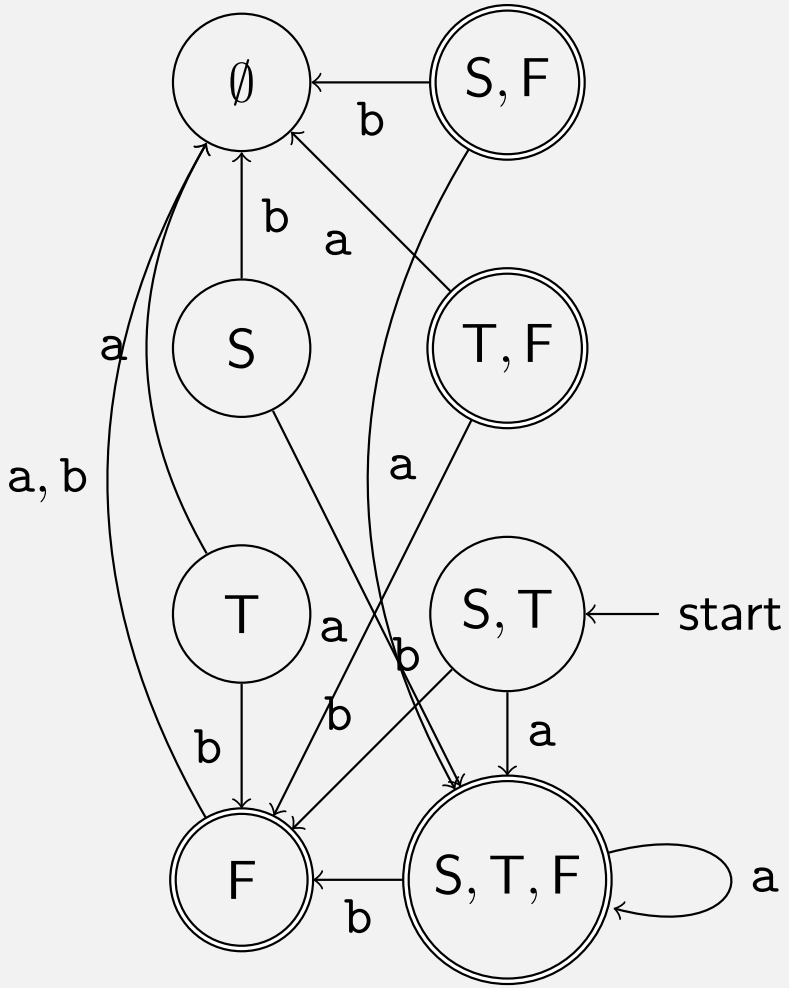
$$d'\,q\,a = \{t \mid r \in q, t \in d\,r\,a\}$$

In summary, we have:

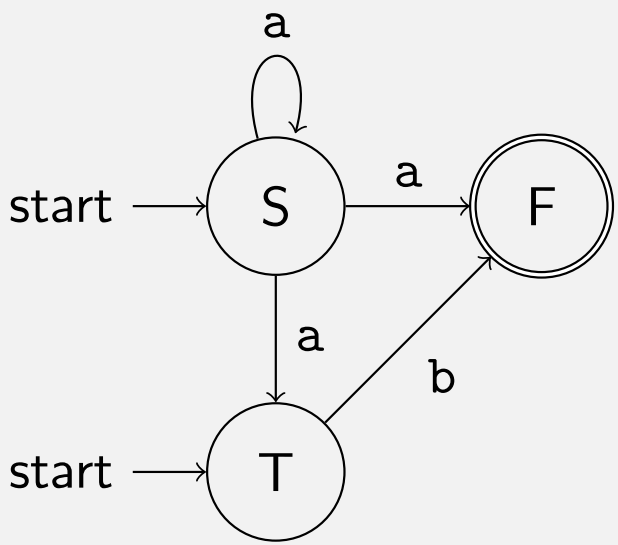
$$(X', Q', d', S', F') = (X, \text{subsets}(Q), d', S, F')$$

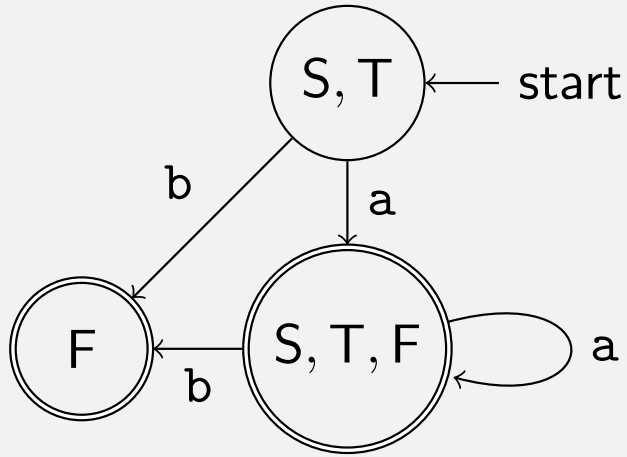
From NFA to DFA – example





From NFA to DFA – example





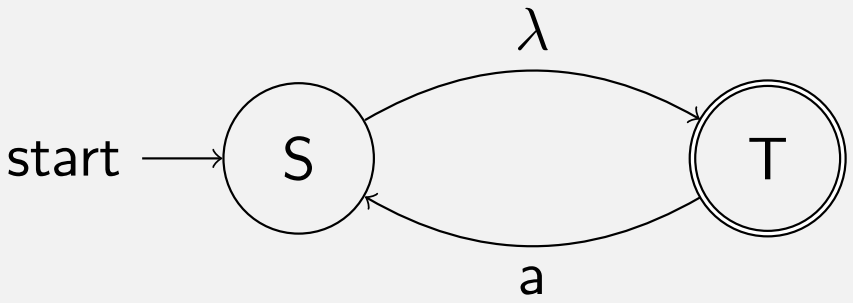
Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

λ-transitions

NFAs can be extended with the possibility of transitions which do **not consume** a character from the string.

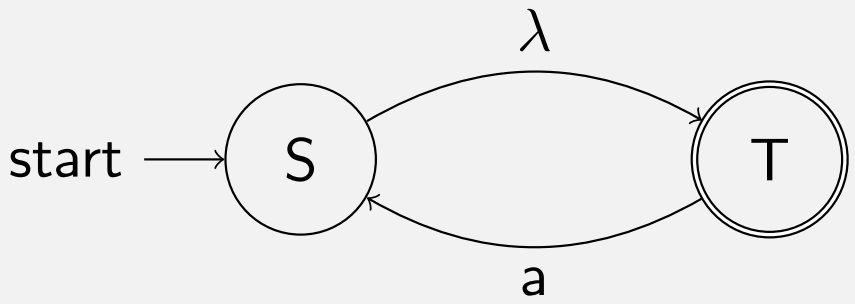
- They are customarily marked with the symbol λ.



λ-transitions

NFAs can be extended with the possibility of transitions which do **not consume** a character from the string.

- They are customarily marked with the symbol λ.



λ-transitions are very useful for proving properties of automata.

λ-transitions

Every NFA (X, Q, d, S, F) with λ-transitions can be made deterministic.

Similar to the powerset construction but:

- Enlarge the set of initial states to those which can be reached by a λ-transition from the original set of states.
- In the transition function d', extend the set of new states which those which can be obtained by one or more additional λ-transitions.

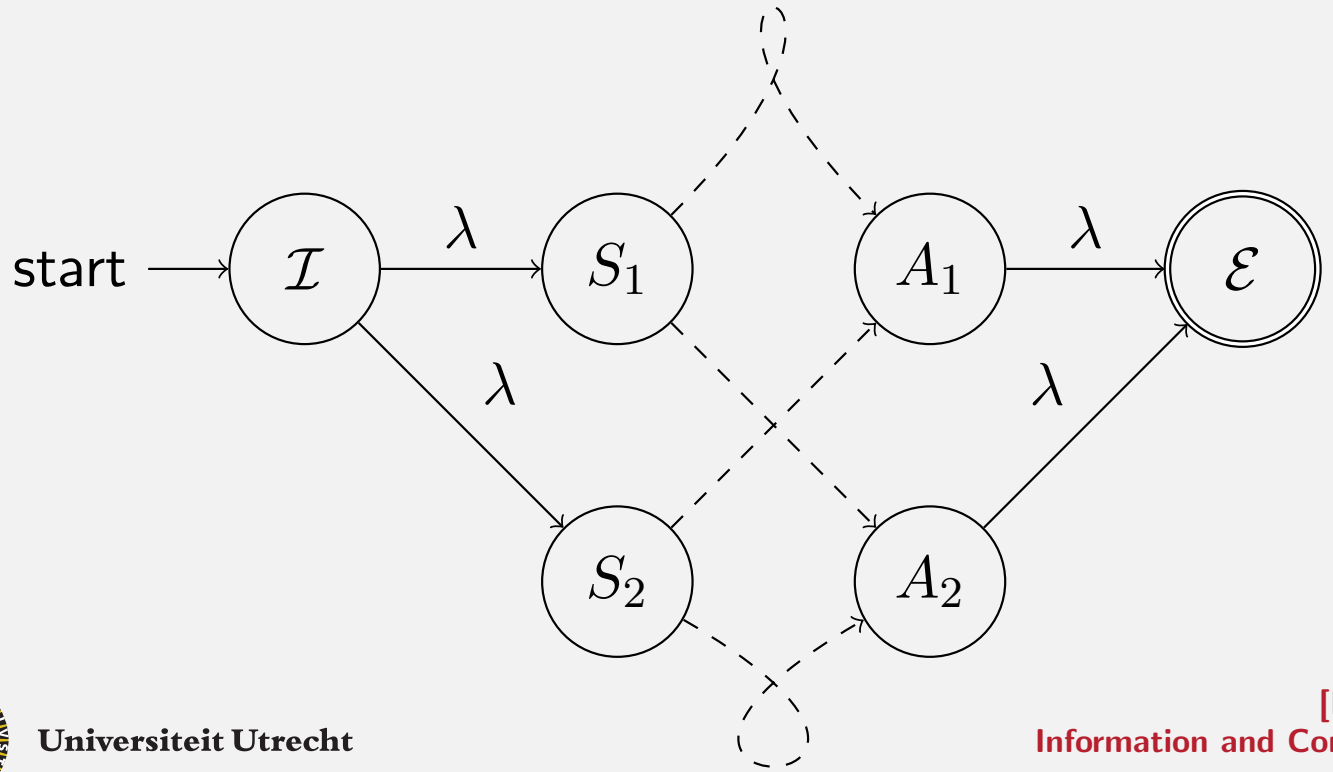
Exercise

Any NFA (X, Q, d, S, F) can be transformed to one which accepts the same language, but with only **one start** state and **at most one accepting** state.

Exercise

Any NFA (X, Q, d, S, F) can be transformed to one which accepts the same language, but with only **one start** state and **at most one accepting** state.

Idea: add two new states I and E and use λ-transitions from and to them.



Exercise

Any NFA (X, Q, d, S, F) can be transformed to one which accepts the same language, but with only **one start** state and **at most one accepting** state.

Formally, the new NFA (X', Q', d', S', F') is defined as:

- The input alphabet is the same, $X' = X$.
- We extend the set of states with two new states, which are supposed to be different from any other, $Q' = Q \cup \{\mathcal{I}, \mathcal{E}\}$.
- We define the transition function d':

$$\begin{array}{lcll} d'\,\mathcal{I}\,\lambda & = & S & \\ d'\,f\,\lambda & = & \{\mathcal{E}\} & \text{if } f \in F \\ d'\,s\,a & = & d\,s\,a & \text{otherwise} \end{array}$$

- Finally, we set the new start and accepting states:

$$S' = \{\mathcal{I}\} \qquad F' = \{\mathcal{E}\}$$

10.5 Closure properties

Summary

Languages defined by NFAs are closed under:

- union,
- concatenation,
- intersection,
- star-closure,
- complement.

Being **closed** under an operation means that if we take languages defined by NFAs, we can build another NFA which recognizes the language defined by applying the operation.

Let's prove them in a constructive* way!

Let's prove them in a constructive* way!

* write the actual NFAs which perform each operation.

A bag full of tricks

As a first step, we can assume several things from the NFAs:

- they work over the same language,
- the names of the states are disjoint,
- they are deterministic,
  - otherwise we apply the powerset construction;
- they have one start state and at most one accepting state,
  - otherwise we apply the previous exercise;
- they do not have λ-transitions.

A bag full of tricks

As a first step, we can assume several things from the NFAs:

- they work over the same language,
- the names of the states are disjoint,
- they are deterministic,
  - otherwise we apply the powerset construction;
- they have one start state and at most one accepting state,
  - otherwise we apply the previous exercise;
- they do not have λ-transitions.

Sometimes it is also useful to consider extreme cases (like having no accepting states, for example) separately from the common case.

Closure under union

Formal statement

- Given two NFAs $A_1 = (X_1, Q_1, d_1, S_1, F_1)$ and $A_2 = (X_2, Q_2, d_2, S_2, F_2)$,
- we can build another NFA A such that $L(A) = L(A_1) \cup L(A_2)$.

Idea of the construction

1. Rename states to remove collision,
2. Start non-deterministically with any of the original start states.

Closure under union

Construction

Assume that the states have been renamed to avoid collision (that is, $Q_1 \cap Q_2 = \emptyset$). Then we build the new automaton $A = (X, Q, d, S, F)$ as follows:

- $X = X_1 \cup X_2$ (we join the alphabets),
- $Q = Q_1 \cup Q_2$ (we join the sets of states),
- The transition function is the pointwise-union of both:

  $$d\ q\ a = d_1\ q\ a \cup d_2\ q\ a$$

- $S = S_1 \cup S_2$ (we may start in any of the automata),
- $F = F_1 \cup F_2$.

Closure under union

This is not the only construction:

- we could have "copied" the automata and add a new start state with λ-transitions to both of them,
- we could have defined the transition function more precisely by noting that always one of d_1 q a or d_2 q a will be empty,
- ...

Closure under union

Proof

In order to make a formal proof, we still need to prove that:

- any word accepted by A_1 or A_2 is accepted by A,
- no words other than those are accepted by A.

Closure under concatenation

Formal statement

- Given two NFAs $A_1 = (X_1, Q_1, d_1, S_1, F_1)$ and $A_2 = (X_2, Q_2, d_2, S_2, F_2)$,
- we can build another NFA A such that $L(A) = L(A_1)\,L(A_2)$.

Reminder

Given two languages L and M, their **concatenation** LM is defined as:

$$LM = \{st \mid s \in L, t \in M\}$$

Informally, one word from L followed by one word from M.

Closure under concatenation

Idea of the construction

Move from accepting states of A_1 to initial states of A_2.

Closure under concatenation

Idea of the construction

Move from accepting states of A_1 to initial states of A_2.

Construction

- $X = X_1 \cup X_2$,
- Assume that Q_1 and Q_2 are disjoint, $Q = Q_1 \cup Q_2$,
- The transition function is extended as follows:

$$d\ q\ a = \begin{cases} d_1\ q\ a & \text{if } a \neq \lambda, \text{ or } q \in Q_1, q \notin F_1 \\ d_1\ q\ a \cup S_2 & \text{if } a = \lambda, q \in Q_1, q \in F_1 \\ d_2\ q\ a & \text{if } q \in Q_2 \end{cases}$$

- $S = S_1$ (you start with the first automaton),
- $F = F_2$ (you must end with the second automaton).

Closure under intersection

Formal statement

- Given two NFAs $A_1 = (X_1, Q_1, d_1, S_1, F_1)$ and $A_2 = (X_2, Q_2, d_2, S_2, F_2)$,
- we can build another NFA A such that $L(A) = L(A_1) \cap L(A_2)$.

That is, the word must be accepted by **both** automata.

Idea of the construction

Keep the state of both automata as a tuple, and accept only if both components are accepting.

Closure under intersection

Construction

We assume that the automata are deterministic (otherwise, apply the powerset construction).

- This will help us defining the transition function.

We build the new automaton $A = (X, Q, d, S, F)$ as follows:

- $X = X_1 \cup X_2$,
- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$,
- $d\ (q_1, q_2)\ a = (d_1\ q_1\ a, d_2\ q_2\ a)$ (apply both transitions),
- $S = (S_1, S_2)$ (only one since they are deterministic),
- $F = F_1 \times F_2$ (both states must be accepting).

Homework (exam practice)

Given an automaton $A = (X, Q, d, S, F)$, build new ones which accept the following languages:

- The star-closure of $L(A)$, $L(A)^*$,
  - Zero or more **repetitions** of a word from $L(A)$;
- The complement of $L(A)$, $\overline{L(A)}$,
  - Words which are **not** in $L(A)$.