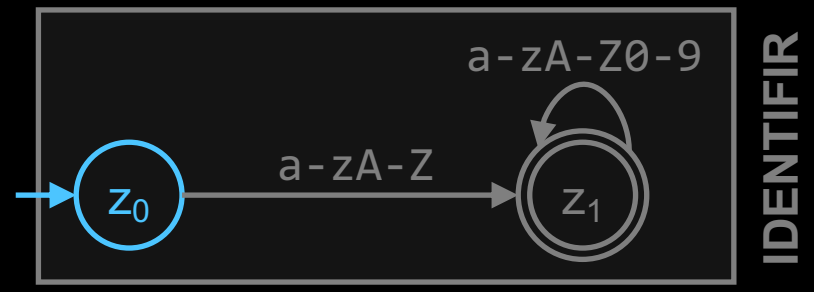
Brainstorming: Beispielhafte Grammatik für Java

```
 1)  <program>     ::= <modifier> <type> <identifier> "{" … "}"

 2)  <modifier>    ::= "public"
 3)  <modifier>    ::= "private"
 4)  <modifier>    ::= "protected"

 5)  <type>        ::= "class"
 6)  <type>        ::= "interface"
 7)  <type>        ::= "enum"

 8)  <identifier>  ::= <letter>
 9)  <identifier>  ::= <letter> <suffix>

10)  <suffix>      ::= <letter>
11)  <suffix>      ::= <digit>

12)  <letter>      ::= "a" | "b" | "c" | … | "x"

13)  <digit>       ::= "0" | "1" | … | "8" | "9"
```

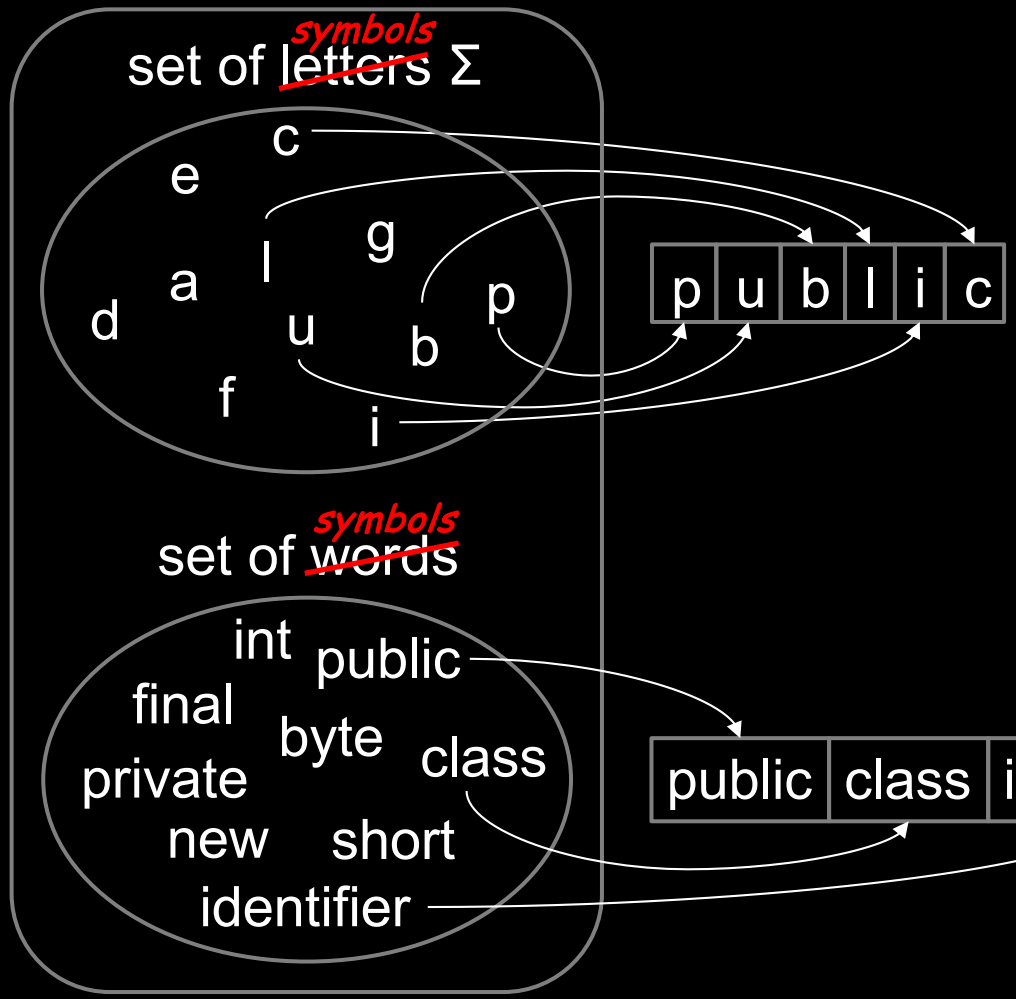
Brainstorming: Beispielhafte Grammatik für Java

```
 1)  <program>     ::= <modifier> <type> <identifier> "{" … "}"

 2)  <modifier>    ::= "public"
 3)  <modifier>    ::= "private"
 4)  <modifier>    ::= "protected"

 5)  <type>        ::= "class"
 6)  <type>        ::= "interface"
 7)  <type>        ::= "enum"

 8)  <identifier>  ::= <letter>
 9)  <identifier>  ::= <letter> <suffix>

10)  <suffix>      ::= <letter>
11)  <suffix>      ::= <digit>

12)  <letter>      ::= "a" | "b" | "c" | … | "x"

13)  <digit>       ::= "0" | "1" | … | "8" | "9"
```



Hatten wir für Bezeichner nicht schon einen endlichen Automaten definiert?

Feststellung: ohne Flexion sind Wörter und Sätze methodisch gleichartig



word ∈ Σ*

language
~~vocabulary~~ ⊆ Σ*

word
~~sentence~~ ∈ Σ*

language ⊆ Σ*

Offene Fragen

1. ~~Nicht jedes Token darf an jeder Stelle stehen. Wie lässt sich das regulieren?~~
2. Sind die von endlichen Automaten akzeptierten Sprachen identisch zu den von Grammatiken erzeugten Sprachen?
3. Kann man die Erstellung eines endlichen Automaten automatisieren?
4. Lassen sich alle NEAs in DEAs umwandeln?

Zeit das gelernte festzuhalten…

concatenation

Let A be a non-empty, finite set.

The set of all strings on A is the set $A^* = \{\langle a_1, \dots, a_n \rangle : n \geq 0 \wedge a_1, \dots, a_n \in A\}$

The operation * is called the Kleene closure.

Concatenation $\circ$ is a two-place operation on A^* defined as $\langle a_1, \dots, a_n \rangle \circ \langle b_1, \dots, b_m \rangle = \langle a_1, \dots, a_n, b_1, \dots, b_m \rangle$

A^* also contains the empty string $\langle\,\rangle$. It is denoted as ε. In particular, $w \circ \varepsilon = w = \varepsilon \circ w$ is valid.

Concatenation is associate, but not commutative.

$$(\alpha \circ \beta) \circ \gamma = a \circ (\beta \circ \gamma)$$
$$\alpha \circ \beta \neq \beta \circ \alpha$$

substrings & atoms

If α is $a_1 \ldots \boldsymbol{b_1} \ldots \boldsymbol{b_n} \ldots a_m$, we call $b_1 \ldots b_n$ a **substring** of α.

A **prefix** of α is an initial substring of α, a **suffix** of α is a final substring of α.

ε is a substring of every string.

α is an **atom** in A^* if $\alpha \neq \varepsilon$ and the **only substrings of α** in A^* **are ε and α** itself.

alphabet

An alphabet is a **non-empty, finite set A** such that every element $a \in A$ is an **atom in A^***.

We call the elements of alphabet A **symbols** or **lexical items**.

An alphabet is **often denoted by Σ**, yet can be given any name.

$B = \{0,1\}$	Says B is an alphabet of two symbols, 0 and 1.
$C = \{a, b, c\}$	Says C is an alphabet of three symbols, a, b and c.

Sometimes **space and comma** are in an alphabet while other times they are **meta symbols** used for descriptions.

symbol / lexical items

A symbol is a character, glyph, mark. It is an **abstract entity** that has **no meaning by itself**, often called uninterpreted. Letters from various alphabets, digits and special characters are the most commonly used symbols.

formal language

A language L in alphabet Σ is a set of strings in Σ^*. We write $L \subseteq \Sigma^*$.

The expression 'aaa' belongs to the language of all expressions that consist only of the letter 'a'.

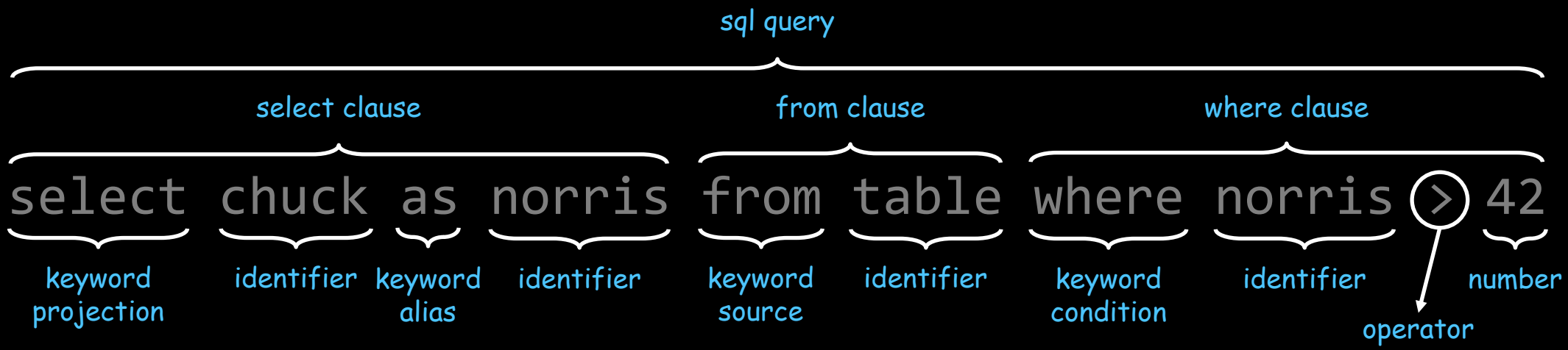
→ word $w \in L$

→ language $L = \{a, aa, aaa, aaaa, \ldots\} = \Sigma^*$

dazu gehört auch Epsilon!

alphabet $\Sigma = \{a\}$

Aufbau formaler Sprachen



Beobachtung: Bei natürlichen Sprachen sind Satzzeichen üblicherweise keine Elemente des Alphabets, obwohl sie in fast jedem korrekten Satz auftreten. Das Alphabet beinhaltet eben nur jene Buchstaben, die benötigt werden, um alle Wörter der Sprache zu bilden.

Formale Sprachen verwenden dagegen häufig einige Sonderzeichen als Abkürzung. Anstelle des größer-Zeichens im oben abgebildeten Beispiel hätte man sich auch auf GT oder „greater than“ festlegen können. Das größer-Zeichen kann demnach ebenfalls als Wort angesehen werden und ist damit Teil des Alphabets.

formal grammar

A formal grammar is specified by a tuple (V_N, V_T, S, P) where

V_N	is a finite set of non-terminal symbols,
V_T	is a finite set of terminal symbols,
$S \in V_N$	is the start symbol,
$P \subset V^* N V^* \times V^*$	is a finite set of production rules,

We write rules $(v, w) \in P$ as $v \to w$. A derivation step has the form $\alpha v \beta \Rightarrow \alpha w \beta$ where $\alpha, \beta \in V^*$.
A derivation has the form $w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$ which we abbreviate as

$$\boldsymbol{w_1 \overset{*}{\Rightarrow} w_n} : \Leftrightarrow \begin{cases} \boldsymbol{w_1 = w_n} \\ \boldsymbol{\exists w \in V^* : w_1 \overset{*}{\Rightarrow} w \wedge w \Rightarrow w_n} \end{cases}$$

The language generated by G is defined as

$$L(G) = \left\{ w \in \Sigma^* \mid \boldsymbol{S \overset{*}{\Rightarrow} w} \right\}.$$

deterministic finite acceptor

A deterministic finite acceptor (DFA) is specified by a tuple $(Q, \Sigma, \delta, q_0, F)$ where

Q	is a finite set of states,
Σ	is a finite alphabet,
$\delta: Q \times \Sigma \to Q$	is the transition function,
$q_0 \in Q$	is the initial state,
$F \subseteq Q$	is the set of final states.

The function $f^*(Q \times \Sigma^*) \to Q$ specifies the state that is reached last when the word w is entered.

$$f^*(q, \varepsilon) = q$$
$$\forall w \in \Sigma^* \; \forall x \in \Sigma: f^*(q, wx) = \delta(f^*(q, w), x)$$

A deterministic finite acceptor $D = (Q, \Sigma, \delta, q_0, F)$ recognizes a language $L \subseteq \Sigma^*$ if

$$L(D) = \{w \in \Sigma^* | f^*(z_0, w) \in F\}.$$