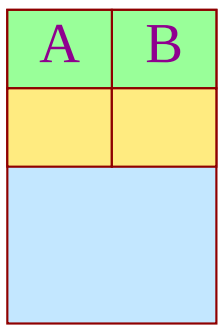
# Idee (Fortsetzung):

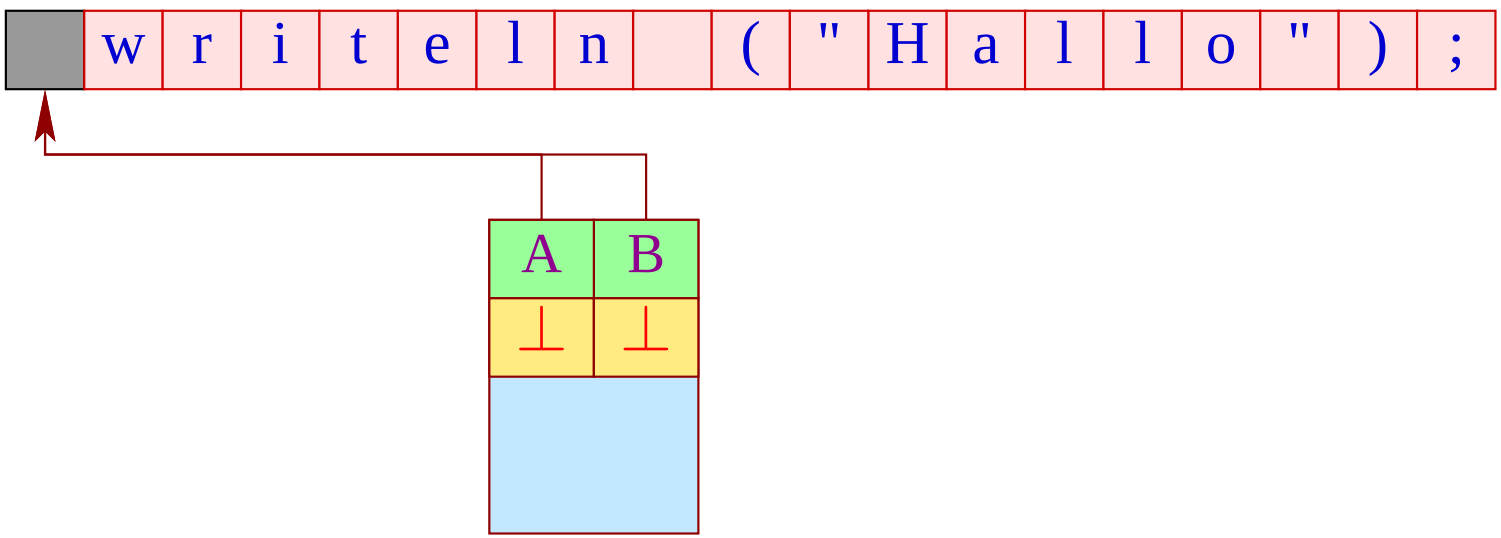
- Der Scanner verwaltet zwei Zeiger $\langle A, B \rangle$ und die zugehörigen Zustände $\langle q_A, q_B \rangle$...
- Der Zeiger $A$ merkt sich die letzte Position in der Eingabe, nach der ein Zustand $q_A \in F$ erreicht wurde;
- Der Zeiger $B$ verfolgt die aktuelle Position.

| s | t | d | o | u | t | . | w | r | i | t | e | l | n |  | ( | " | H | a | l | l | o | " | ) | ; |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



# Idee (Fortsetzung):

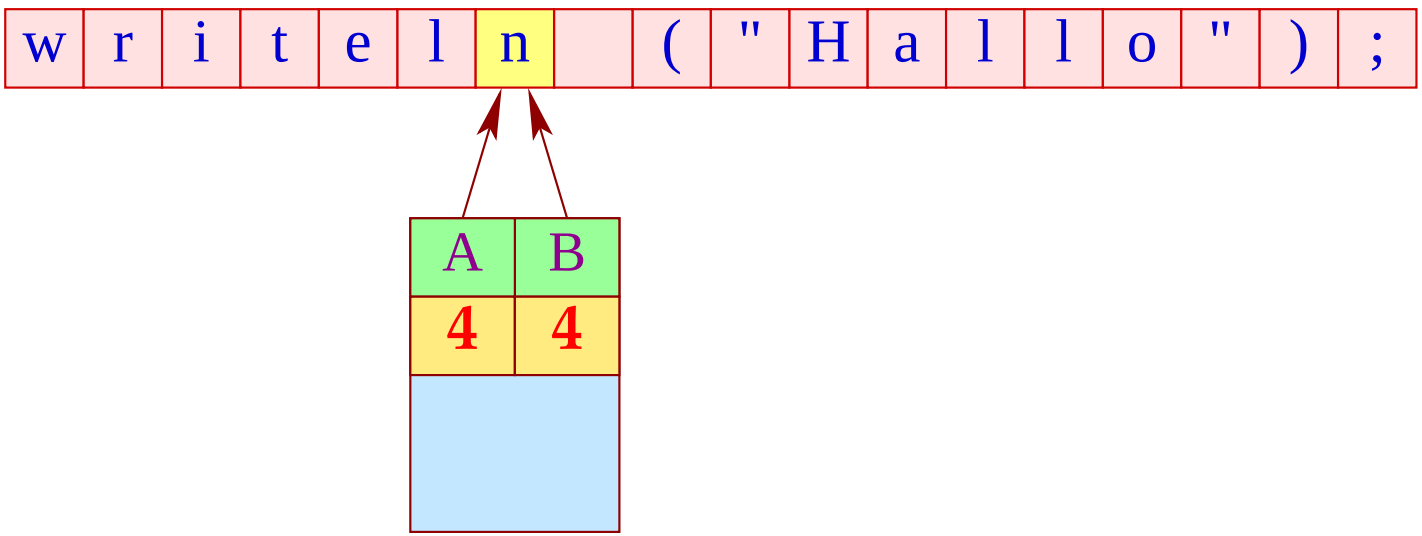
- Der Scanner verwaltet zwei Zeiger $\langle A, B \rangle$ und die zugehörigen Zustände $\langle q_A, q_B \rangle$...
- Der Zeiger $A$ merkt sich die letzte Position in der Eingabe, nach der ein Zustand $q_A \in F$ erreicht wurde;
- Der Zeiger $B$ verfolgt die aktuelle Position.



# Idee (Fortsetzung):

- Ist der aktuelle Zustand $q_B = \emptyset$ , geben wir Eingabe bis zur Postion $A$ aus und setzen:

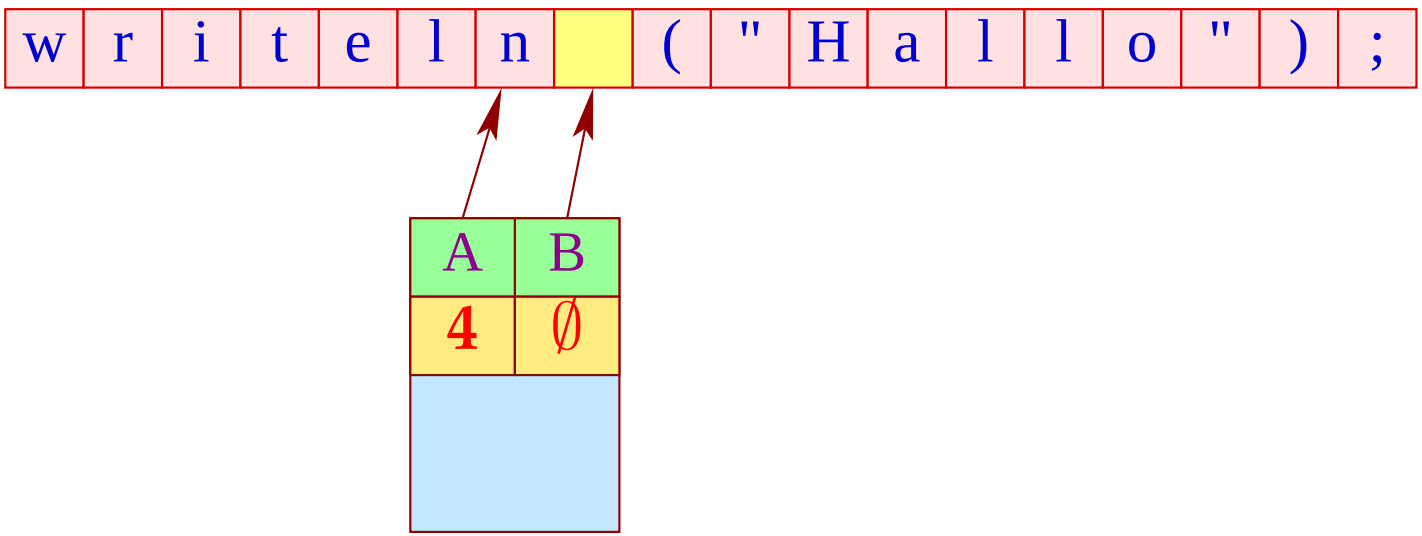
$$B := A; \quad A := \perp;$$
$$q_B := q_0; \quad q_A := \perp$$



# Idee (Fortsetzung):

- Ist der aktuelle Zustand $q_B = \emptyset$ , geben wir Eingabe bis zur Postion $A$ aus und setzen:

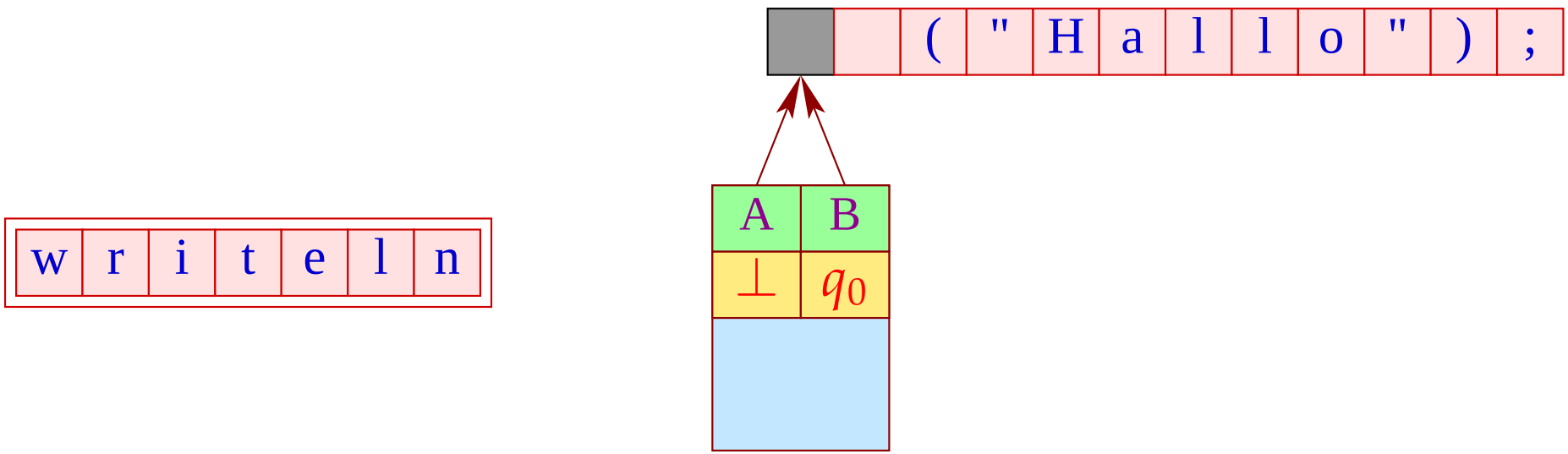
$$B := A; \quad A := \perp;$$
$$q_B := q_0; \quad q_A := \perp$$



# Idee (Fortsetzung):

- Ist der aktuelle Zustand $q_B = \emptyset$ , geben wir Eingabe bis zur Postion $A$ aus und setzen:

$$B := A; \quad A := \perp;$$
$$q_B := q_0; \quad q_A := \perp$$



# Erweiterung: Zustände

- Gelegentlich ist es nützlich, unterschiedliche Scanner-Zustände zu unterscheiden.
- In unterschiedlichen Zuständen sollen verschiedene Tokenklassen erkannt werden können.
- In Abhängigkeit der gelesenen Tokens kann der Scanner-Zustand geändert werden ;-)

## Beispiel: Kommentare

Innerhalb eines Kommentars werden Identifier, Konstanten, Kommentare, ... nicht erkannt ;-)

## Eingabe (verallgemeinert): eine Menge von Regeln:

$\langle$`state`$\rangle$ { $e_1$ { $\texttt{action}_1$ $\texttt{yybegin(state}_1\texttt{);}$ }
$e_2$ { $\texttt{action}_2$ $\texttt{yybegin(state}_2\texttt{);}$ }
...
$e_k$ { $\texttt{action}_k$ $\texttt{yybegin(state}_k\texttt{);}$ }
}

- Der Aufruf `yybegin (state`$_i$`);` setzt den Zustand auf `state`$_i$.
- Der Startzustand ist (z.B. bei JFlex) `YYINITIAL`.

## ... im Beispiel:

```
⟨YYINITIAL⟩    "/*"    { yybegin(COMMENT); }
⟨COMMENT⟩    { "*/"    { yybegin(YYINITIAL); }
               . | \n    {  }
             }
```

## Bemerkungen:

- "`.`" matcht alle Zeichen ungleich "`\n`".
- Für jeden Zustand generieren wir den entsprechenden Scanner.
- Die Methode `yybegin (STATE);` schaltet zwischen den verschiedenen Scannern um.
- Kommentare könnte man auch direkt mithilfe einer geeigneten Token-Klasse implementieren. Deren Beschreibung ist aber ungleich komplizierter :-)
- Scanner-Zustände sind insbesondere nützlich bei der Implementierung von Präprozessoren, die in einen Text eingestreute Spezifikationen expandieren sollen.

## 1.4 Implementierung von DFAs

### Aufgaben:

- Implementiere die Übergangsfunktion $\delta : Q \times \Sigma \to Q$
- Implementiere eine Klassifizierung $r : Q \to \mathbb{N}$

## 1.4 Implementierung von DFAs

### Aufgaben:

- Implementiere die Übergangsfunktion $\delta : Q \times \Sigma \to Q$
- Implementiere eine Klassifizierung $r : Q \to \mathbb{N}$

### Probleme:

- Die Anzahl der Zustände kann sehr groß sein :-(
- Das Alphabet kann sehr groß sein: z.B. Unicode :-((

# Reduzierung der Anzahl der Zustände

## Idee: Minimierung

- Identifiziere Zustände, die sich im Hinblick auf eine Klassifizierung $r$ gleich verhalten :-)
- Sei $A = (Q, \Sigma, \delta, \{q_0\}, r)$ ein DFA mit Klassifizierung. Wir definieren auf den Zuständen eine Äquivalenzrelation durch:

$$p \equiv_r q \quad \text{gdw.} \quad \forall\, w \in \Sigma^* \;:\; r(\delta(p,w)) = r(\delta(q,w))$$

- Die neuen Zustände sind Äquivalenzklassen der alten Zustände :-)

| | |
|---|---|
| Zustände | $[q]_r, q \in Q$ |
| Anfangszustand | $[q_0]_r$ |
| Klassifizierung | $r([q]_r) = r(q)$ |
| Übergangsfunktion | $\delta([p]_r, a) = [\delta(p,a)]_r$ |

## Problem: Wie berechnet man $\equiv_r$ ?

## Idee:

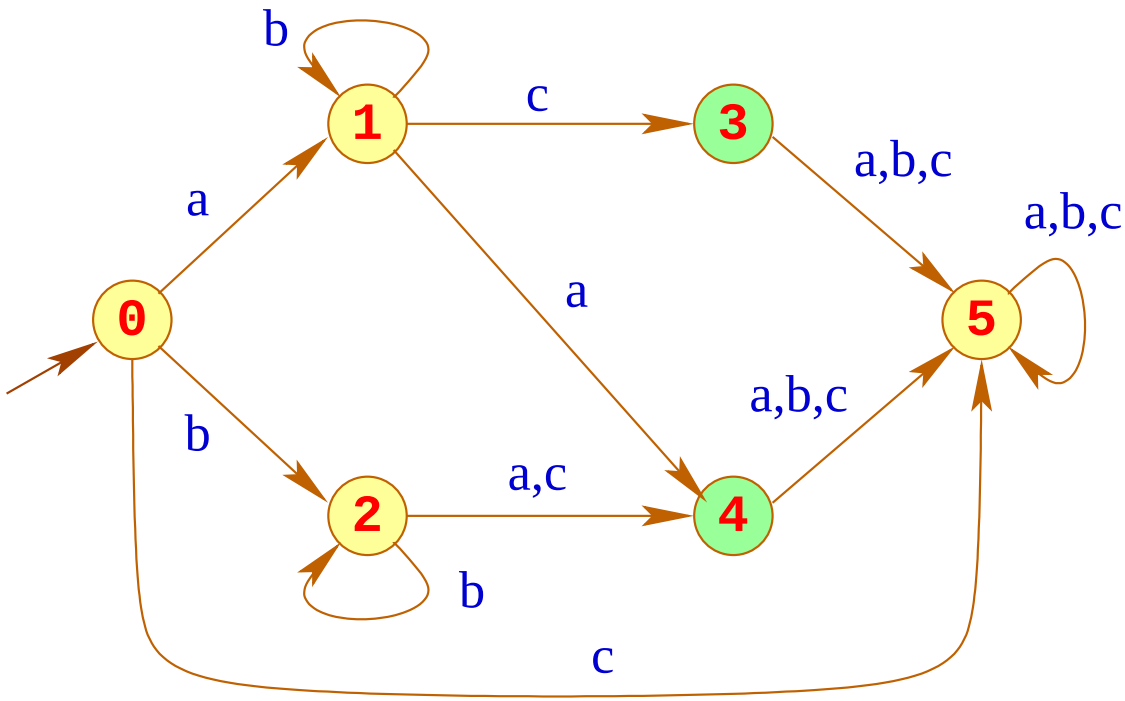
- Wir nehmen an, maximal viel sei äquivalent :-)

  Wir starten mit der Partition:

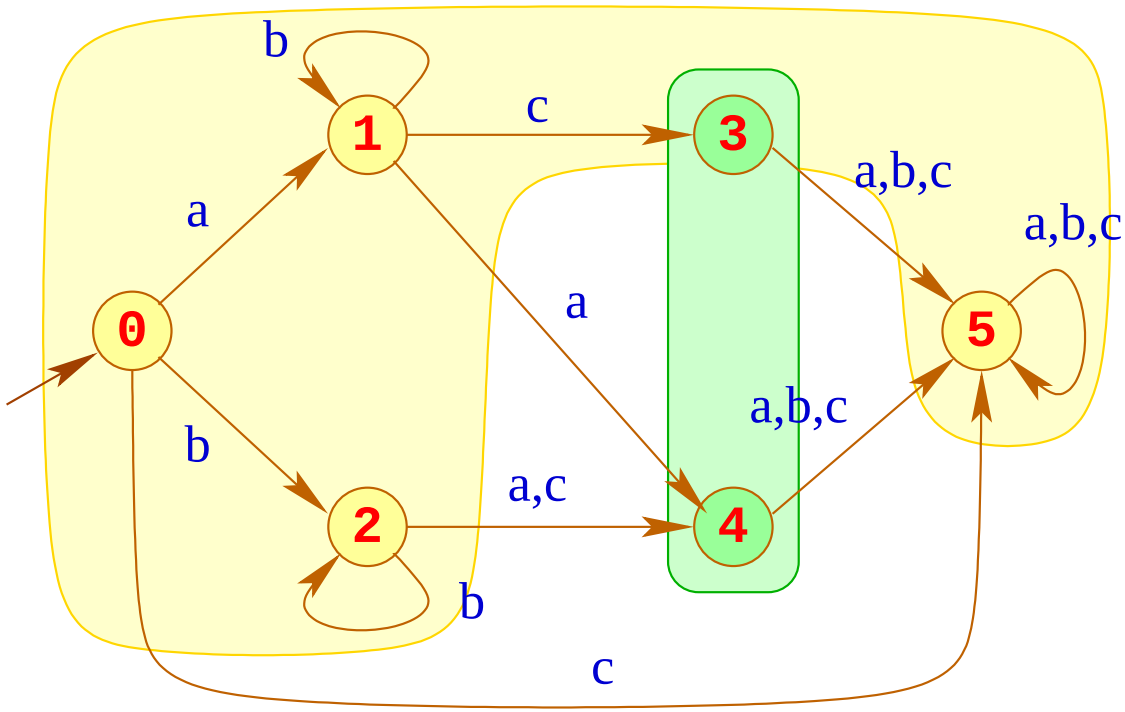
$$\overline{Q} = \{r^{-1}(i) \neq \emptyset \mid i \in \mathbb{N}\}$$

- Finden wir in $\bar{q} \in \overline{Q}$ Zustände $p_1, p_2$ sodass $\delta(p_1, a)$ und $\delta(p_2, a)$ in verschiedenen Äquivalenzklassen liegen (für irgend ein $a$ ), müssen wir $\bar{q}$ aufteilen ...

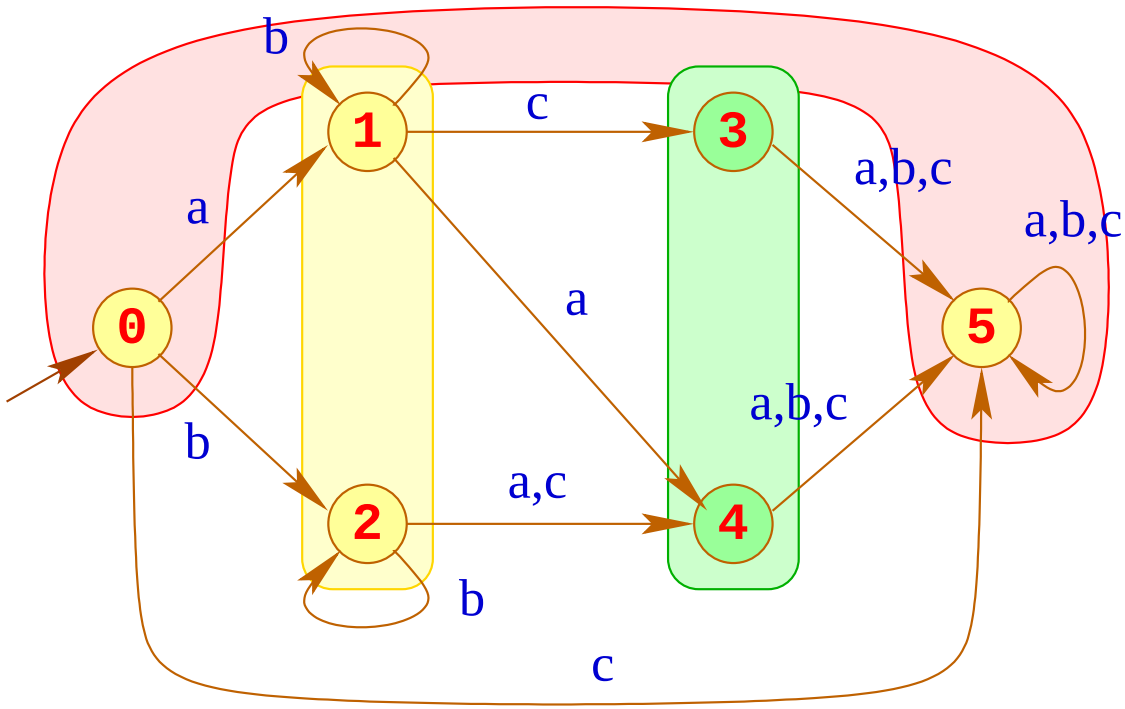
# Beispiel:



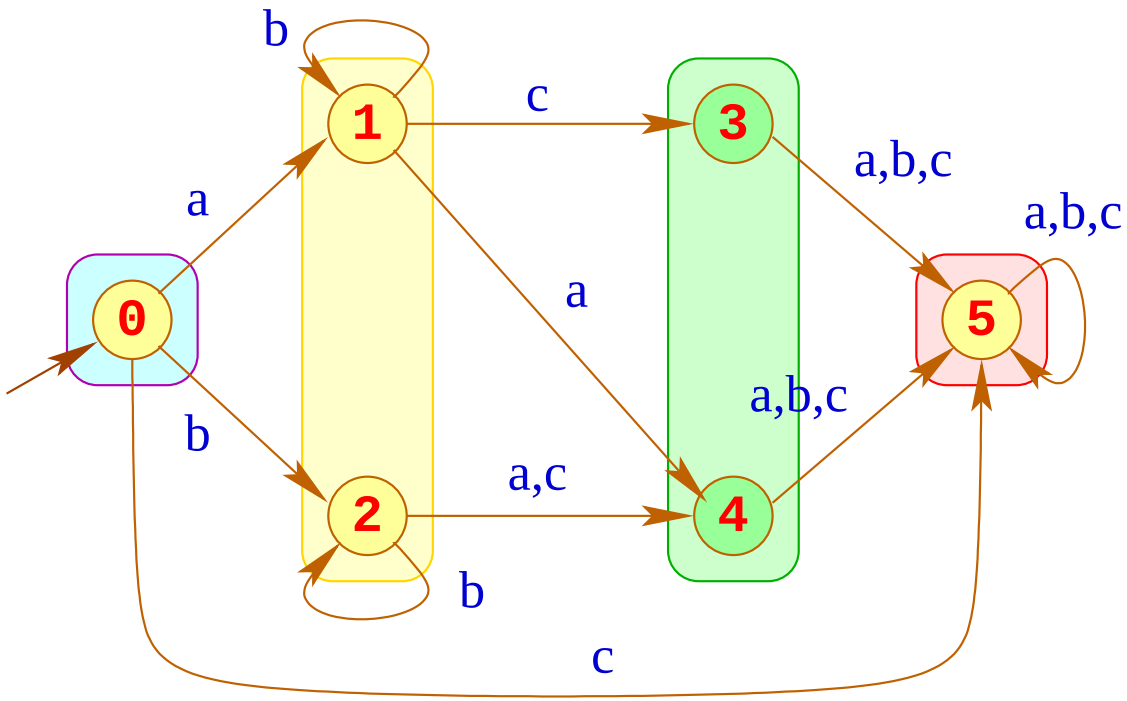
# Beispiel:



# Beispiel:



# Beispiel:



# Bemerkungen:

- Das Verfahren liefert die gröbste Partition $\overline{Q}$, die mit $r$ und $\delta$ verträglich ist, d.h. für $\bar{q} \in \overline{Q}$,

  (1) $p_1, p_2 \in \bar{q} \implies r(p_1) = r(p_2)$

  (2) $p_1, p_2 \in \bar{q} \implies \delta(p_1, a), \delta(p_2, a)$ gehören zur gleichen Klasse

- Der Ergebnis-Automat ist der eindeutig bestimmte minimale Automat für $\mathcal{L}(A)$ ;-)
- Eine naive Implementierung erfordert Laufzeit $\mathcal{O}(n^2)$.

  Eine raffinierte Verwaltung der Partition liefert ein Verfahren mit Laufzeit $\mathcal{O}(n \cdot \log(n))$.

Anil Nerode , Cornell University, Ittaca

John E. Hopcroft, Cornell University, Ittaca

# Reduzierung der Tabellengröße

## Problem:

- Die Tabelle für $\delta$ wird mit Paaren $(q, a)$ indiziert.
- Sie enthält eine Spalte für jedes $a \in \Sigma$.
- Das Alphabet $\Sigma$ umfasst i.a. ASCII, evt. aber ganz Unicode :-(

# 1. Idee:

- Bei großen Alphabeten wird man in der Spezifikation i.a. nicht einzelne Zeichen auflisten, sondern Zeichenklassen benutzen :-)
- Lege Spalten nicht für einzelne Zeichen sondern für Klassen von Zeichen an, die sich gleich verhalten.

# Beispiel:

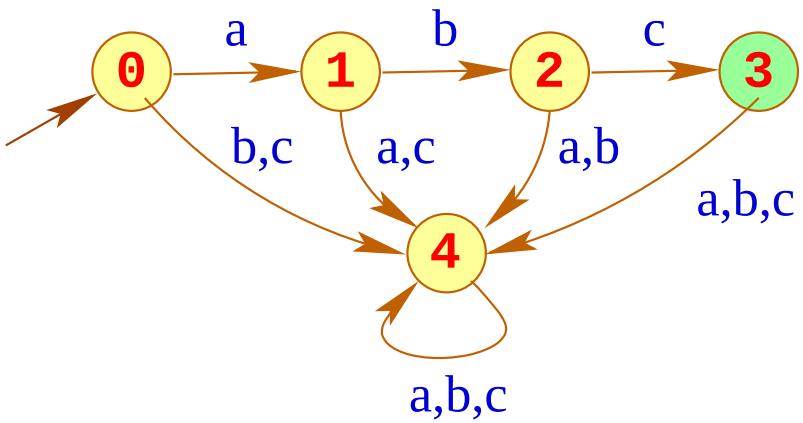
```
le = [a-zA-Z_\$]
ledi = [a-zA-Z_\$0-9]
Id = {le} {ledi}*
```

- Der Automat soll deterministisch sein.
- Sind die Klassen der Spezifikation nicht disjunkt, teilt man sie darum in Unterklassen auf, hier in die Klassen `[a-zA-Z_\$]` und `[0-9]` :-)

## 2. Idee:

- Finden wir, dass mehrere (Unter-) Klassen der Spezifikation in der Spalte übereinstimmen, können wir sie nachträglich wieder vereinigen :-)
- Wir können weitere Methoden der Tabellen-Komprimierung anwenden, z.B. Zeilenverschiebung (Row Displacement) ...

## Beispiel:



## ... die zugehörige Tabelle (transponiert):

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| *a* | 1 | 4 | 4 | 4 | 4 |
| *b* | 4 | 2 | 4 | 4 | 4 |
| *c* | 4 | 4 | 3 | 4 | 4 |

## Beobachtung:

- Viele Einträge in der Tabelle sind gleich einem Wert Default (hier: 4)
- Diesen Wert brauchen wir nicht zu repräsentieren :-)

## ... die zugehörige Tabelle (transponiert):

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| *a* | 1 | | | | |
| *b* | | 2 | | | |
| *c* | | | 3 | | |

## Beobachtung:

- Viele Einträge in der Tabelle sind gleich einem Wert Default (hier: 4)
- Diesen Wert brauchen wir nicht zu repräsentieren :-)
- Dann legen wir einfach mehrere (transponierte) Spalten übereinander :-))

# ... im Beispiel:

| | 0 | 1 | 2 |
|---|---|---|---|
| A | 1 | 2 | 3 |
| valid | *a* | *b* | *c* |

- Feld valid teilt mit, für welches Element aus $\Sigma$ der Eintrag gilt :-)
- Achtung: I.a. werden die Spalten nicht so perfekt übereinander passen!

  Dann verschieben wir sie so lange, bis die jeweils nächste in die bisherigen Löcher hineinpasst.
- Darum müssen wir ein zusätzliches Feld displacement verwalten, in dem wir uns die Verschiebung merken ;-)

Ein Feld-Zugriff $\delta(j,a)$ wird dann so realisiert:

$$
\begin{array}{lcl}
\delta(j,a) & = & \mathsf{let}\quad d \quad = \quad \mathsf{displacement}[a] \\
 & & \mathsf{in}\ \ \mathsf{if}\ \ (\mathsf{valid}[d+j] \equiv a) \\
 & & \qquad \mathsf{then}\ \ \mathsf{A}[d+j] \\
 & & \qquad \mathsf{else}\ \ \mathsf{Default} \\
 & & \mathsf{end}
\end{array}
$$

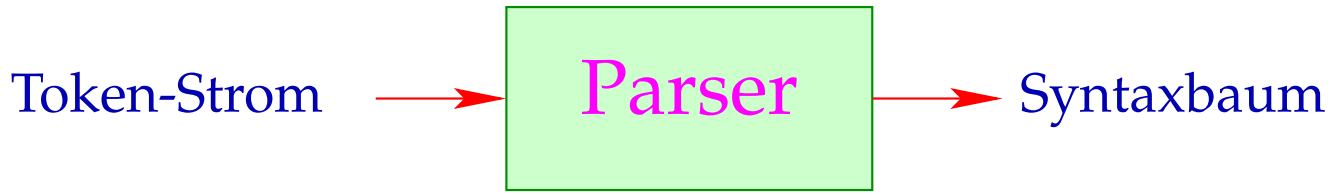
Ein Feld-Zugriff $\delta(j,a)$ wird dann so realisiert:

$$
\begin{array}{lll}
\delta(j,a) & = & \mathsf{let}\quad d \quad = \quad \mathsf{displacement}[a] \\
 & & \mathsf{in}\ \ \mathsf{if}\ \ (\mathsf{valid}[d+j] \equiv a) \\
 & & \qquad \mathsf{then}\ \ \mathsf{A}[d+j] \\
 & & \qquad \mathsf{else}\ \ \mathsf{Default} \\
 & & \mathsf{end}
\end{array}
$$

## Diskussion:

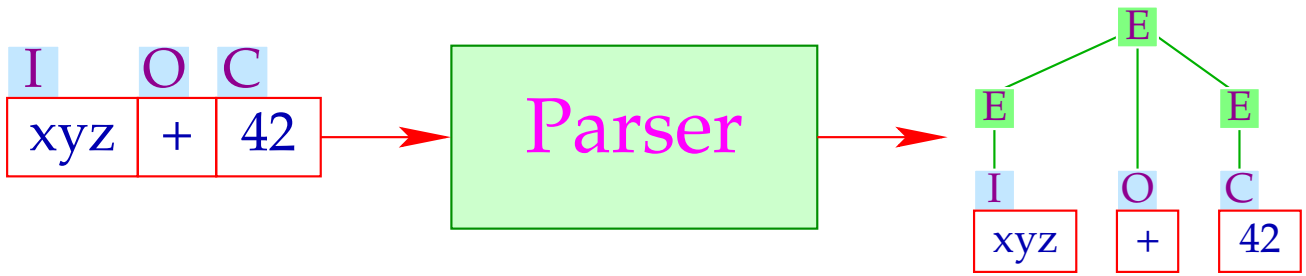
- Die Tabellen werden i.a. erheblich kleiner.
- Dafür werden Tabellenzugriffe etwas teurer.

# 2 Die syntaktische Analyse




- Die syntaktische Analyse versucht, Tokens zu größeren Programmeinheiten zusammen zu fassen.

# 2 Die syntaktische Analyse




- Die syntaktische Analyse versucht, Tokens zu größeren Programmeinheiten zusammen zu fassen.
- Solche Einheiten können sein:
  - → Ausdrücke;
  - → Statements;
  - → bedingte Verzweigungen;
  - → Schleifen; ...

# Diskussion:

Auch Parser werden i.a. nicht von Hand programmiert, sondern aus einer Spezifikation generiert:

# Diskussion:

Auch Parser werden i.a. nicht von Hand programmiert, sondern aus einer Spezifikation generiert:


**Spezifikation der hierarchischen Struktur:** kontextfreie Grammatiken;

**Generierte Implementierung:** Kellerautomaten + X :-)

## 2.1 Grundlagen: Kontextfreie Grammatiken

- Programme einer Programmiersprache können unbeschränkt viele Tokens enthalten, aber nur endlich viele Token-Klassen :-)
- Als endliches Terminal-Alphabet $T$ wählen wir darum die Menge der Token-Klassen.
- Die Schachtelung von Programm-Konstrukten lässt sich elegant mit Hilfe von kontextfreien Grammatiken beschreiben ...

## 2.1 Grundlagen: Kontextfreie Grammatiken

- Programme einer Programmiersprache können unbeschränkt viele Tokens enthalten, aber nur endlich viele Token-Klassen :-)
- Als endliches Terminal-Alphabet $T$ wählen wir darum die Menge der Token-Klassen.
- Die Schachtelung von Programm-Konstrukten lässt sich elegant mit Hilfe von kontextfreien Grammatiken beschreiben ...

Eine kontextfreie Grammatik (CFG) ist ein 4-Tupel $G = (N, T, P, S)$ , wobei:

- $N$ die Menge der Nichtterminale,
- $T$ die Menge der Terminale,
- $P$ die Menge der Produktionen oder Regeln, und
- $S \in N$ das Startsymbol ist.

Noam Chomsky, MIT (Guru)

John Backus, IBM (Erfinder von Fortran)

Die Regeln kontextfreier Grammatiken sind von der Form:

$$A \to \alpha \quad \text{mit} \quad A \in N\,, \;\; \alpha \in (N \cup T)^*$$

Die Regeln kontextfreier Grammatiken sind von der Form:

$$A \to \alpha \quad \text{mit} \quad A \in N\,,\ \ \alpha \in (N \cup T)^*$$

## Beispiel:

$$\begin{aligned} S &\to a\,S\,b \\ S &\to \epsilon \end{aligned}$$

Spezifizierte Sprache: $\{a^n b^n \mid n \geq 0\}$

Die Regeln kontextfreier Grammatiken sind von der Form:

$$A \to \alpha \quad \text{mit} \quad A \in N\,, \quad \alpha \in (N \cup T)^*$$

## Beispiel:

$$\begin{aligned} S &\to a\,S\,b \\ S &\to \epsilon \end{aligned}$$

Spezifizierte Sprache: $\{a^n b^n \mid n \geq 0\}$

## Konventionen:

- In Beispielen ist die Spezifikation der Nichtterminale und Terminale i.a. implizit:
  - → Nichtterminale sind: $A, B, C, \ldots, \langle \mathsf{exp} \rangle, \langle \mathsf{stmt} \rangle, \ldots$;
  - → Terminale sind: $a, b, c, \ldots, \mathsf{int}, \mathsf{name}, \ldots$;

# Weitere Beispiele:

$$
\begin{array}{lcl}
S & \rightarrow & \langle \text{stmt} \rangle \\
\langle \text{stmt} \rangle & \rightarrow & \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{rexp} \rangle ; \\
\langle \text{if} \rangle & \rightarrow & \text{if ( } \langle \text{rexp} \rangle \text{ ) } \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle \\
\langle \text{while} \rangle & \rightarrow & \text{while ( } \langle \text{rexp} \rangle \text{ ) } \langle \text{stmt} \rangle \\
\langle \text{rexp} \rangle & \rightarrow & \text{int} \mid \langle \text{lexp} \rangle \mid \langle \text{lexp} \rangle = \langle \text{rexp} \rangle \mid \ldots \\
\langle \text{lexp} \rangle & \rightarrow & \text{name} \mid \ldots
\end{array}
$$

# Weitere Beispiele:

$$
\begin{array}{lcl}
S & \rightarrow & \langle \text{stmt} \rangle \\
\langle \text{stmt} \rangle & \rightarrow & \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{rexp} \rangle ; \\
\langle \text{if} \rangle & \rightarrow & \text{if } ( \langle \text{rexp} \rangle ) \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle \\
\langle \text{while} \rangle & \rightarrow & \text{while } ( \langle \text{rexp} \rangle ) \langle \text{stmt} \rangle \\
\langle \text{rexp} \rangle & \rightarrow & \text{int} \mid \langle \text{lexp} \rangle \mid \langle \text{lexp} \rangle = \langle \text{rexp} \rangle \mid \ldots \\
\langle \text{lexp} \rangle & \rightarrow & \text{name} \mid \ldots
\end{array}
$$

# Weitere Konventionen:

- Für jedes Nichtterminal sammeln wir die rechten Regelseiten und listen sie gemeinsam auf :-)
- Die $j$-te Regel für $A$ können wir durch das Paar $(A, j)$ bezeichnen ($j \geq 0$).

# Weitere Grammatiken:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $E$ | $\rightarrow$ | $E+E$ | $\mid$ | $E*E$ | $\mid$ | $(\ E\ )$ | $\mid$ | name | $\mid$ | int |

| | | | | | | |
|---|---|---|---|---|---|---|
| $E$ | $\rightarrow$ | $E+T$ | $\mid$ | $T$ | | |
| $T$ | $\rightarrow$ | $T*F$ | $\mid$ | $F$ | | |
| $F$ | $\rightarrow$ | $(\ E\ )$ | $\mid$ | name | $\mid$ | int |

Die beiden Grammatiken beschreiben die gleiche Sprache ;-)

# Weitere Grammatiken:

$$E \rightarrow E+E\,^{0} \mid E*E\,^{1} \mid (\ E\ )\,^{2} \mid \mathsf{name}\,^{3} \mid \mathsf{int}\,^{4}$$

$$\begin{aligned} E &\rightarrow E+T\,^{0} \mid T\,^{1} \\ T &\rightarrow T*F\,^{0} \mid F\,^{1} \\ F &\rightarrow (\ E\ )\,^{0} \mid \mathsf{name}\,^{1} \mid \mathsf{int}\,^{2} \end{aligned}$$

Die beiden Grammatiken beschreiben die gleiche Sprache ;-)

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

## ... im letzten Beispiel:

$$\underline{E}$$

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

## ... im letzten Beispiel:

$$\underline{E} \quad \rightarrow \quad \underline{E} + T$$

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

## ... im letzten Beispiel:

$$\begin{aligned} \underline{E} &\rightarrow \underline{E} + T \\ &\rightarrow \underline{T} + T \end{aligned}$$

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

## ... im letzten Beispiel:

$$\begin{aligned} \underline{E} &\rightarrow \underline{E} + T \\ &\rightarrow \underline{T} + T \\ &\rightarrow T * \underline{F} + T \end{aligned}$$

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

## ... im letzten Beispiel:

$$
\begin{aligned}
\underline{E} &\rightarrow \underline{E} + T \\
&\rightarrow \underline{T} + T \\
&\rightarrow T * \underline{F} + T \\
&\rightarrow \underline{T} * \mathsf{int} + T
\end{aligned}
$$

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

## ... im letzten Beispiel:

$$
\begin{aligned}
\underline{E} &\rightarrow \underline{E} + T \\
&\rightarrow \underline{T} + T \\
&\rightarrow T * \underline{F} + T \\
&\rightarrow \underline{T} * \mathsf{int} + T \\
&\rightarrow \underline{F} * \mathsf{int} + T
\end{aligned}
$$

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

## ... im letzten Beispiel:

$$
\begin{aligned}
\underline{E} &\rightarrow \underline{E} + T \\
&\rightarrow \underline{T} + T \\
&\rightarrow T * \underline{F} + T \\
&\rightarrow \underline{T} * \mathsf{int} + T \\
&\rightarrow \underline{F} * \mathsf{int} + T \\
&\rightarrow \mathsf{name} * \mathsf{int} + \underline{T}
\end{aligned}
$$

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

## ... im letzten Beispiel:

$$
\begin{aligned}
\underline{E} &\rightarrow \underline{E} + T \\
&\rightarrow \underline{T} + T \\
&\rightarrow T * \underline{F} + T \\
&\rightarrow \underline{T} * \mathsf{int} + T \\
&\rightarrow \underline{F} * \mathsf{int} + T \\
&\rightarrow \mathsf{name} * \mathsf{int} + \underline{T} \\
&\rightarrow \mathsf{name} * \mathsf{int} + \underline{F}
\end{aligned}
$$

Grammatiken sind Wortersetzungssysteme.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch Ableitung.

## ... im letzten Beispiel:

$$
\begin{aligned}
\underline{E} &\rightarrow \underline{E} + T \\
&\rightarrow \underline{T} + T \\
&\rightarrow T * \underline{F} + T \\
&\rightarrow \underline{T} * \mathsf{int} + T \\
&\rightarrow \underline{F} * \mathsf{int} + T \\
&\rightarrow \mathsf{name} * \mathsf{int} + \underline{T} \\
&\rightarrow \mathsf{name} * \mathsf{int} + \underline{F} \\
&\rightarrow \mathsf{name} * \mathsf{int} + \mathsf{int}
\end{aligned}
$$

Formal ist $\rightarrow$ eine Relation auf Wörtern über $V = N \cup T$ , wobei

$$\alpha \rightarrow \alpha' \quad \text{gdw.} \quad \alpha = \alpha_1\ A\ \alpha_2 \ \wedge\ \alpha' = \alpha_1\ \beta\ \alpha_2 \ \text{ für ein } \ A \rightarrow \beta \in P$$

Den reflexiven und transitiven Abschluss von $\rightarrow$ schreiben wir: $\rightarrow^*$ :-)

Formal ist $\rightarrow$ eine Relation auf Wörtern über $V = N \cup T$, wobei

$$\alpha \rightarrow \alpha' \quad \text{gdw.} \quad \alpha = \alpha_1\ A\ \alpha_2 \ \wedge\ \alpha' = \alpha_1\ \beta\ \alpha_2 \ \text{ für ein } \ A \rightarrow \beta \in P$$

Den reflexiven und transitiven Abschluss von $\rightarrow$ schreiben wir: $\rightarrow^*$ :-)

## Bemerkungen:

- Die Relation $\rightarrow$ hängt von der Grammatik ab ;-)
- Eine Folge von Ersetzungsschritten: $\alpha_0 \rightarrow \ldots \rightarrow \alpha_m$ heißt Ableitung.
- In jedem Schritt einer Ableitung können wir:
  - eine Stelle auswählen, wo wir ersetzen wollen, sowie
  - eine Regel, wie wir ersetzen wollen.
- Die von $G$ spezifizierte Sprache ist:

$$\mathcal{L}(G) = \{w \in T^* \mid S \rightarrow^* w\}$$