

# Compilerbau I

## Kompaktmerkzettel (Muster)

von Alexander Köster  
Student der Universität Siegen, CB1 2020  
Letzte Aktualisierung: 25. Juli 2020

Dieser Kurs der theoretischen Informatik befasst sich mit den Grundlagen und Bauteilen eines Compilers.

*Einzig für Lernzwecke erstellt.*

*Nicht geeignet als Klausurhilfe.*

*Das Erstellen einer eigenen Klausurhilfe führt zu einem besonders guten Lerneffekt.*

*Dieses Dokument sollte nur als Orientierung oder Vergleich dienen.*

*Jeder sollte seine Klausurhilfe individuell auf seinen Lernstand und seine eigenen Probleme anpassen,  
gut bekannte Dinge auslassen und schlecht merkbare Dinge hinzufügen.*

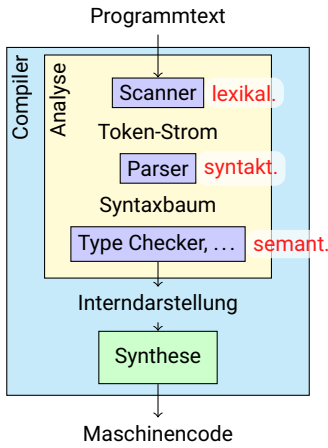
*Dieses Dokument beinhaltet viel mehr, als für eine tatsächliche Klausur durchschnittlich benötigt wird.*

*Für einen guten Ansatz, welche Inhalte man braucht, sollte sich an der Probeklausur orientiert werden.*

© study.woalk.de

Vervielfältigung ohne ausdrückliche Erlaubnis des Autors außerhalb der originalen Website untersagt.

# 0 Einführung



## 1 Analyse-Phase

### 1.1 Lexikalische Analyse

- **Token-Klassen:** *identifier, constants, operators, keywords*
- **Sieber:** Vorverarbeitung klassifizierter Tokens (*whitespace, Pragmas, Symbol-Tabelle*)
- Scanner und Sieber werden i. A. in einer Komponente zusammengefasst (Scanner+Aktion).
- Scanner i. A. nicht von Hand programmiert, sondern aus Spezifikation autom. generiert

#### 1.1.1 Grundlagen: Reguläre Ausdrücke

- Die Menge der Textabschnitte einer Token-Klasse ist i. A. eine **reguläre Sprache**.
- Um Klammern zu sparen:  $* > \cdot > |$
- **Def.:** Für den regulären Ausdruck  $e \in \mathcal{E}_\Sigma$  ist die **spezifizierte Sprache**  $\llbracket e \rrbracket \subseteq \Sigma^*$  ind. def. durch:
  - $\llbracket \epsilon \rrbracket = \{\epsilon\}$
  - $\llbracket a \rrbracket = \{a\}$
  - $\llbracket e^* \rrbracket = (\llbracket e \rrbracket)^*$ , wobei für bel. Sprachen (Mengen)  $L$  gilt:  $L^* := \{w_1 \dots w_k \mid k \geq 0, w_i \in L \forall i \in \underline{k}\}$
  - $\llbracket e_1 | e_2 \rrbracket = \llbracket e_1 \rrbracket \cup \llbracket e_2 \rrbracket$
  - $\llbracket e_1 \cdot e_2 \rrbracket = \llbracket e_1 \rrbracket \cdot \llbracket e_2 \rrbracket$ , wobei für bel. Sprachen (Mengen)  $L_1, L_2$  gilt:  $L_1 \cdot L_2 := \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$

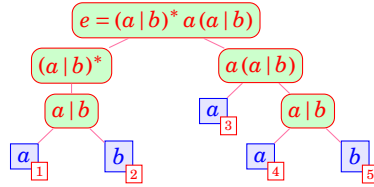
**Bsp.:**  $\llbracket ab^*a \rrbracket = \{ab^n a \mid n \geq 0\}$

#### 1.1.2 Grundlagen: Endliche Automaten

- **$\epsilon$ -NFA** (nicht-deterministischer endl. Automat mit  $\epsilon$ -Übergängen):  $A = (Q, \Sigma, \delta, I, F)$  mit:
  - $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$  Übergangs-Relation
 Andere endl. Automaten:
  - $\triangleright$  Gibt es keine  $\epsilon$ -Übergänge, ist  $A$  ein **NFA**.
  - $\triangleright$  Ist  $\delta : Q \times \Sigma \rightarrow Q$  eine Funktion und  $|I| = 1$ , heißt  $A$  deterministisch (**DFA**).
- Die Menge aller akzeptierten Worte ist die **akzeptierte Sprache** vom endl. Automat  $A$ :  $\mathcal{L}(A) = \{w \in \Sigma^* \mid \exists i \in I, f \in F : (i, w, f) \in \delta^*\}$
- **Transitiver Abschluss**  $\delta^*$  von  $\delta$  ist die kleinste Menge  $\delta'$  mit, für alle  $x \in \Sigma \cup \{\epsilon\}, w \in \Sigma^*$ :
  - $\triangleright (p, \epsilon, q) \in \delta'$
  - $\triangleright$  Wenn  $(p, x, p_1) \in \delta' \wedge (p_1, w, q) \in \delta' \Rightarrow (p, xw, q) \in \delta'$ $\delta^*$  beschreibt für je 2 Zustände, mit welchen Wörtern man von einem zum anderen kommt.
- **Satz:**  $\forall e \in \mathcal{E}_\Sigma$  kann (in lin. Zeit) ein  $\epsilon$ -NFA konstruiert werden, der die Sprache  $\llbracket e \rrbracket$  akzeptiert.

### Konstruktion des $\epsilon$ -NFA für $e \in \mathcal{E}_\Sigma$ :

#### 1. Nummeriere die Blätter. Bsp.:



Alternativ: Verwende  $\text{num} : \mathcal{E}_\Sigma \rightarrow \mathcal{E}_{\mathbb{N} \times \Sigma}$ .

**Bsp.:**  $\text{num}((a|b)^* a(a|b)) = ((1, a) | (2, b))^* (3, a) ((4, a) | (5, b))$

#### 2. Zustände: $\bullet r, r \bullet$ für alle Knoten $r$ von $e$

Anfangszustand:  $\bullet e$ , Endzustand:  $e \bullet$

#### Übergangsrelation:

- $\triangleright$  Für Blätter  $r = [i, x] : (\bullet r, x, r \bullet) \in \delta$
- $\triangleright$  Für Knoten der Form  $r = r_1 | r_2$ :
  - $(\bullet r, \epsilon, \bullet r_1) \in \delta$
  - $(\bullet r, \epsilon, \bullet r_2) \in \delta$
  - $(r_1 \bullet, \epsilon, r \bullet) \in \delta$
  - $(r_2 \bullet, \epsilon, r \bullet) \in \delta$
- $\triangleright$  Für Knoten der Form  $r = r_1 \cdot r_2$ :
  - $(\bullet r, \epsilon, \bullet r_1) \in \delta$
  - $(r_1 \bullet, \epsilon, \bullet r_2) \in \delta$
  - $(r_2 \bullet, \epsilon, r \bullet) \in \delta$
- $\triangleright$  Für Knoten der Form  $r = r_1^*$ :
  - $(\bullet r, \epsilon, r \bullet) \in \delta$
  - $(\bullet r, \epsilon, \bullet r_1) \in \delta$
  - $(r_1 \bullet, \epsilon, \bullet r) \in \delta$
  - $(r_1 \bullet, \epsilon, r \bullet) \in \delta$
- $\triangleright$  Für Knoten der Form  $r = r_1^?$ :
  - $(\bullet r, \epsilon, r \bullet) \in \delta$
  - $(\bullet r, \epsilon, \bullet r_1) \in \delta$
  - $(r_1 \bullet, \epsilon, r \bullet) \in \delta$

### • Konstruktion des NFA ohne $\epsilon$ -Übergänge:

Zunächst einige Definitionen:

- $\triangleright \text{empty}[r] = \text{true} \Leftrightarrow \epsilon \in \llbracket r \rrbracket$ 
  - Für Blätter  $r = [i, x]$  ist  $\text{empty}[r] := (x \equiv \epsilon)$
  - $\text{empty}[r_1 | r_2] := \text{empty}[r_1] \vee \text{empty}[r_2]$
  - $\text{empty}[r_1 \cdot r_2] := \text{empty}[r_1] \wedge \text{empty}[r_2]$
  - $\text{empty}[r_1^*] := \text{true}$
  - $\text{empty}[r_1^?] := \text{true}$
- $\triangleright \text{first}[r] := \{i \text{ in } r \mid (\bullet r, \epsilon, \bullet [i, x]) \in \delta^*, x \neq \epsilon\}$ 
  - Für Blätter  $r = [i, x]$  ist  $\text{first}[r] := \{i \mid x \neq \epsilon\}$
  - $\text{first}[r_1 | r_2] := \text{first}[r_1] \cup \text{first}[r_2]$
  - $\text{first}[r_1 \cdot r_2] := \begin{cases} \text{first}[r_1] \cup \text{first}[r_2] & \text{empty}[r_1] = \text{true} \\ \text{first}[r_1] & \text{empty}[r_1] = \text{false} \end{cases}$
  - $\text{first}[r_1^*] := \text{first}[r_1]$
  - $\text{first}[r_1^?] := \text{first}[r_1]$
- $\triangleright \text{next}[r] := \{i \in \mathbb{N} \mid (r \bullet, \epsilon, \bullet [i, x]) \in \delta^*, x \neq \epsilon\}$ 

Hier wird induktiv andersherum, also von oben, definiert:

  - Für die Wurzel  $e$  ist  $\text{next}[e] := \emptyset$ .
  - Ist der übergeordnete Knoten  $r = \dots r_1 | r_2$ , dann:
    - $\text{next}[r_1] := \text{next}[r]$ ,
    - $\text{next}[r_2] := \text{next}[r]$
  - $\dots r_1 \cdot r_2$ , dann:
    - $\text{next}[r_1] := \begin{cases} \text{first}[r_2] \cup \text{next}[r] & \text{empty}[r_2] = \text{true} \\ \text{first}[r_2] & \text{empty}[r_2] = \text{false} \end{cases}$

$\text{next}[r_2] := \text{next}[r]$

$\dots r_1^*$ , dann:

$\text{next}[r_1] := \text{first}[r_1] \cup \text{next}[r]$

$\dots r_1^?$ , dann:

$\text{next}[r_1] := \text{next}[r]$

$\triangleright \text{last}[r] := \{i \text{ in } r \mid ([i, x] \bullet, \epsilon, r \bullet) \in \delta^*, x \neq \epsilon\}$

- Für Blätter  $r = [i, x]$  ist  $\text{last}[r] := \{i \mid x \neq \epsilon\}$
- $\text{last}[r_1 | r_2] := \text{last}[r_1] \cup \text{last}[r_2]$
- $\text{last}[r_1 \cdot r_2] := \begin{cases} \text{last}[r_1] \cup \text{last}[r_2] & \text{empty}[r_2] = \text{true} \\ \text{last}[r_2] & \text{empty}[r_2] = \text{false} \end{cases}$
- $\text{last}[r_1^*] := \text{last}[r_1]$
- $\text{last}[r_1^?] := \text{last}[r_1]$

Der **Berry-Sethi-/Glushkow-NFA**  $A_e$  ist dann:

**Zustände:**  $\{q_0\} \cup \{i \in \mathbb{N} \mid i \text{ in Blatt } [i, x], x \neq \epsilon\}$

**Startzustand:**  $I := \{q_0\}$

**Endzustände:** Falls  $\text{empty}[e] = \text{false}$ , dann  $F := \text{last}[e]$ ; sonst  $F := \{q_0\} \cup \text{last}[e]$ .

(wobei  $\cup$  die disjunkte Vereinigung ist)

**Übergänge:**  $\forall i, i' \in \mathbb{N} :$

- $\triangleright (q_0, x, i) \in \delta$ , falls  $i \in \text{first}[e]$  und das Blatt  $i$  mit  $x$  beschriftet ist.
- $\triangleright (i, y, i') \in \delta$ , falls  $i' \in \text{next}[\llbracket [i, x] \rrbracket]$ , das Blatt  $i$  mit  $x$  und  $i'$  mit  $y$  beschriftet ist.

• **Satz:** Zu jedem NFA  $A = (Q, \Sigma, \delta, I, F)$  kann ein DFA  $\mathcal{P}(A) = (\text{Pot}(Q), \Sigma, \delta_{\mathcal{P}}, \{i\}, F_{\mathcal{P}})$  konstruiert werden mit  $\mathcal{L}(A) = \mathcal{L}(\mathcal{P}(A))$ .

**Teilmengen-/Potenzmengenkonstruktion:**

**Zustände:**  $\text{Pot}(Q)$  (alle Teilmengen von  $Q$ )

**Startzustand:**  $i := I \in \text{Pot}(Q)$

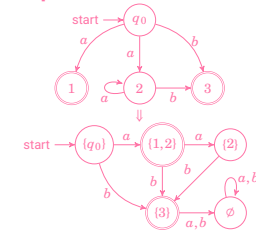
**Endzustände:**  $F_{\mathcal{P}} := \{Q' \subseteq Q \mid Q' \cap F \neq \emptyset\}$

**Übergangsfunktion:**

$\delta_{\mathcal{P}}(Q', a) := \{q \in Q \mid \exists p \in Q' : (p, a, q) \in \delta\}$

Um die Anzahl Zustände mgl. klein zu halten, beginnen wir mit  $Q_{\mathcal{P}} = \{i\}$  und fügen weitere Zustände nur nach Bedarf hinzu.

**Bsp.:**



•  $\Rightarrow$  **Satz:**  $\forall e \in \mathcal{E}_\Sigma$  kann ein DFA  $A = \mathcal{P}(A_e)$  konstruiert werden mit  $\mathcal{L}(A) = \llbracket e \rrbracket$ .

#### 1.1.3 Design eines Scanners

• **Eingabe:** eine Menge von Regeln:

$e_1 \quad \{\text{action}_1\}$   
 $e_2 \quad \{\text{action}_2\}$   
 $\vdots \quad \vdots$   
 $e_k \quad \{\text{action}_k\}$

mit  $k \in \mathbb{N}, e_i \in \mathcal{E}_\Sigma$  und  $\epsilon \notin \llbracket e_i \rrbracket \forall i \in \underline{k}$ .

**Ausgabe:** ein Programm, das

- $\triangleright$  von der Eingabe ein **maximales Präfix**  $w$  liest, das  $e_1 | \dots | e_k$  erfüllt,
- $\triangleright$  das minimale  $i$  ermittelt mit  $w \in \llbracket e_i \rrbracket$ ,
- $\triangleright$  für  $w$  dann  $\text{action}_i$  ausführt.

• **Konstruktion:**

1. Konstruiere den DFA  $\mathcal{P}(A_e)$  (Pot.-A. des Berry-Sethi-NFAs) zu  $e := e_1 | e_2 | \dots | e_k$ .

2. Definiere die Mengen:

- $\triangleright F_1 := \{q \in F \mid q \cap \text{last}[e_1] \neq \emptyset\}$
- $\triangleright F_2 := \{q \in (F \setminus F_1) \mid q \cap \text{last}[e_2] \neq \emptyset\}$
- $\vdots$

$$F_k := \{q \in (F \setminus (F_1 \cup \dots \cup F_{k-1})) \mid q \cap \text{last}[e_k] \neq \emptyset\}$$

Für Eingabe  $w$  gilt:  $\delta_p^*(q_0, w) \in F_i \Leftrightarrow$  der Scanner für  $w$  action <sub>$i$</sub>  ausführen soll.

• **Tokenizing-/Reps' Maximal-Munch-Algorithmus**

INPUT: String  $w \in \Sigma^+$

$s := 1; k := 1$

FORALL  $q \in Q_p, i \in |w| + 1$  DO

$\text{fehlversuch}[q, i] := \text{false}$

  WHILE true DO

$q := q_0; p := \perp; S := \{(q_0, k)\}$

    WHILE  $k \leq |w|$  AND  $\delta(q, w[k]) \neq \emptyset$  DO

      IF  $\text{fehlversuch}[q, k] = \text{true}$  THEN

        BREAK

$q := \delta(q, w[k]); k := k + 1$

      IF  $q \in F$  THEN

$p := q; j := k; S := \emptyset$

      ELSE  $S := S \cup \{(q, k)\}$

    ENDWHILE

  IF  $p = \perp$  THEN RETURN (failure)

  FORALL  $(q, k) \in S$  DO

$\text{fehlversuch}[q, k] := \text{true}$

  let  $i$  such that  $p \in F_i$

  write( $w[s, j - 1]$ )

  action <sub>$i$</sub> ()

$s := j; k := j$

  IF  $j = |w| + 1$  THEN RETURN

ENDWHILE

Der **Maximal-Munch-Algorithmus** benötigt nur  $\mathcal{O}(|Q| \cdot |w|)$ , Tokenizing braucht  $\mathcal{O}(|w|^2)$ .

- In unterschiedlichen **Scannerzuständen** können verschiedene Token-Klassen erkannt werden, **Bsp.:** Kommentare.

Eingabe: eine Menge von Regeln:

```

Bsp.:
<YYINITIAL> {
  /* {yybegin(COMMENT)}
  }
  <COMMENT> {
    /* {yybegin(YYINITIAL)}
    .\n { }
  }

```

1.1.4 Implementierung von DFAs

- Die **Klassifizierungsfunktion**  $r: Q \rightarrow \mathbb{N}$  ist für

$$\mathcal{P}(A_e) \text{ definiert durch } r(q) := \begin{cases} 0 & q \notin F \\ i & q \in F_i \end{cases}$$

- **Def.** Äquivalenzrel. auf Zuständen:  $p \equiv_r q \Leftrightarrow \forall w \in \Sigma^*: r(\delta^*(p, w)) = r(\delta^*(q, w))$

**Konstruktion des minimierten DFA** für  $\mathcal{L}(A)$ :

Zustände:  $[q]_r \quad \forall q \in Q$

Anfangszustand:  $[q_0]_r$

Klassifizierung:  $r([q]_r) := r(q)$

Übergangsfunktion:  $\delta([p]_r, a) := [\delta(p, a)]_r$

Dazu: **Konstruktion der Äquiv.-klassen:**

1.  $\bar{Q} := \{r^{-1}(i) \mid i \in \mathbb{N}, r^{-1}(i) \neq \emptyset\}$   
(Partition von  $Q$  in die Fasern von  $r$ , **dabei ist die Faser**  $r^{-1}(i) := \{q \in Q \mid r(q) = i\}$ , d. h. Zustände gleicher Klassifizierung.)
2. Teste  $\forall \bar{q} \in \bar{Q} \forall p_1, p_2 \in \bar{q} \forall a \in \Sigma$ , ob  $r(\delta(p_1, a)) \neq r(\delta(p_2, a))$ . In dem Fall muss  $\bar{q}$  entsprechend aufgeteilt werden.

Naive Implementierung:  $\mathcal{O}(n^2)$ , Optimierung auf  $\mathcal{O}(n \cdot \log n)$  möglich.

- **Konstruktion des minimierten DFA**, alt. Alg.:

1. Starte Tabelle aller Zustandspaare (treppenförmig, horiz. letzten Zustand weglassen, vert. 1. Zustand weglassen)
2. Markiere  $\{q, q'\}$ , falls:
  - ▷ Erkennungsäq. (GTI):  $q \in F, q' \notin F$
  - ▷  $r$ -Äquivalenz:  $r(q) \neq r(q')$
3.  $\forall$  unmarkierten Paare: teste  $\forall a \in \Sigma$ , ob  $\{\delta(q, a), \delta(q', a)\}$  bereits markiert. Ja  $\Rightarrow$  markiere  $\{q, q'\}$ .

Bsp.:

2					
3	1	1			
4	1	1			
5	1	1	1	1	
6	2	3	1	1	1
	1	2	3	4	5

1. Schritt

2. Schritt

3. Schritt

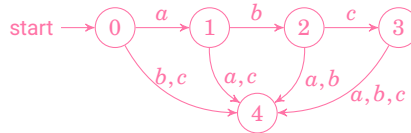
4. Wdh., bis keine Änd. der Tabelle mehr.

5.  $\forall$  jetzt noch unmark. Paare gilt:  $q \equiv q'$

6. Verschmelze je äquivalente Zustands-paare, vereinige Übergänge der beiden.

- Speichern der Übergangsfunktion als Tabelle indiziert mit  $Q \times \Sigma$ .

Bsp.:



$\delta$	0	1	2	3	4
a	1	4	4	4	4
b	4	2	4	4	4
c	4	4	3	4	4

- Reduktion der Tabellengröße

▷ durch Zusammenfassung von Zeichen in Zeichenklassen, **Bsp.:**  $[a-zA-Z\_]$

▷ häufigsten Wert nicht speichern & durch „default“ ersetzen, **Bsp.:**

$\delta$	0	1	2	3	4
a	1				
b		2			
c			3		

▷ Übereinanderlegen der Zeilen, **Bsp.:**

$\delta$	0	1	2
A	1	2	3
valid	a	b	c

Passen die Zeilen nicht direkt übereinander, schieben wir diese nach rechts, bis sie passen, und merken uns die Verschiebung als displacement. Je weniger zusätzliche Spalten nötig sind, desto besser. **Bsp.:**

$\delta$	0	1	2	3	4
a	1		2	1	
b	3	2		0	

$\delta$	0	1	2	3	4	5
a			1		2	1
b	3	2		0		

displacement(a) = 2,  
displacement(b) = 0

1.2 Syntaktische Analyse

- Zusammenfassen von Tokens mit **Parser** zu größeren Programmeinheiten, **Bsp.:** Ausdrücke, Statements, if, Schleifen, ...

- Auch Parser werden i. A. nicht von Hand progr., sondern aus Spezifikation generiert.

- Spezifikation: kontextfreie Grammatiken mit Terminalalphabet = Menge der Token-Klassen  
Generierte Impl.: Kellerautomaten +X

1.2.1 Grundlagen: Kontextfreie Grammatiken

- **Def.:** **kontextfr. Grammatik:**  $G = (N, T, P, S)$ 
  - ▷  $T$ : endl. Menge der Terminale
  - ▷  $P$ : Menge der Produktionen/Regeln der Form  $A \rightarrow \alpha$  mit  $A \in N, \alpha \in (N \cup T)^*$

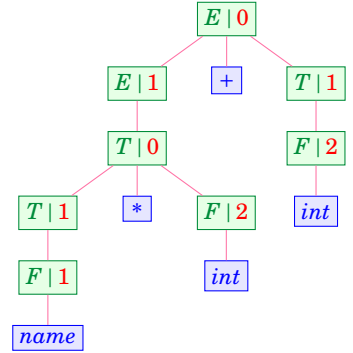
- Sammle & nummeriere  $\forall$  Nichtterminal  $A$  die rechten Seiten:  $(A, j)$  ist  $j$ -te Regel von  $A$ .

```

Bsp. KfGI: E → E + T0 | T1
          T → T * F0 | F1
          F → (E)0 | name1 | int2

```

• **Ableitungsbaum:** **Bsp.:**  $t :=$



innere Knoten: Regel-Anwendungen,  
Wurzel: Regel-Anwendung für  $A$ ,  
Blätter: Terminale oder  $\epsilon$

Kinder eines Knotens  $(B, i)$  entsprechen der rechten Seite der  $i$ -ten Regel von  $B$ .

- **Links-/Rechts-Ableitungen** sind solche, bei denen man stets das linke bzw. rechte Vorkommen eines Nichtterminals ersetzt.

▷ entsprechen links-rechts- bzw. rechts-links-preorder-DFS-Durchlauf durch den Ableitungsbaum.

▷ **Reverse Rechts-Abl.** entsprechen links-rechts-postorder-DFS-Durchlauf.

- Konkatenation der Blätter eines Ableitungsbaums  $t$  heißt  $\text{yield}(t)$ .

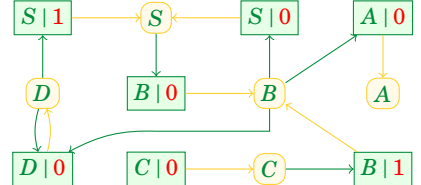
**Bsp.:**  $\text{yield}(t) = \text{name} * \text{int} + \text{int}$

- Die Grammatik  $G$  heißt **eindeutig**, falls es  $\forall w \in T^*$  max. einen Ableitungsbaum  $t$  von  $S$  gibt mit  $\text{yield}(t) = w$ .

1.2.2 Überflüssige Nichtterminale & Regeln

- **Def.:**  $A \in N$  heißt...
  - ▷ **produktiv**, falls  $A \rightarrow^* w$  für ein  $w \in T^*$ .
  - ▷ **erreichbar**, falls  $S \xrightarrow{*} \alpha A \beta$  für geeignete  $\alpha, \beta \in (T \cup N)^*$ .

- Für Produktivität: **And-Or-Graph**. **Bsp.:**



**And-Knoten:** ■ (nummerierte) Regeln

**Or-Knoten:** • Nichtterminale

**Kanten:** ↗  $((B, i), B)$  für alle Regeln  $(B, i)$ ,  
↘  $(A, (B, i))$  falls  $(B, i) \equiv B \rightarrow \alpha_1 A \alpha_2$

**Algorithmus für Produktivität:**

**Subset**  $\langle N \rangle$  **result** :=  $\emptyset$  (d. h. **result**  $\subseteq N$ )  
**int** count  $\langle P \rangle$

**Subset**  $\langle P \rangle$  **rhs**  $\langle N \rangle$

FORALL  $A \in N$  DO **rhs**  $\langle A \rangle := \emptyset$

FORALL  $(A, i) \in P$  DO

  count  $\langle (A, i) \rangle := 0$

**init**  $\langle A, i \rangle$

  ↳ gehe rechte Seite der Produktion  $(A, i)$  durch. Für jedes versch. Nichtt.  $X$  darin: count  $\langle (A, i) \rangle ++$  und **rhs**  $\langle X \rangle$ .add  $\langle (A, i) \rangle$ .

ENDFORALL

**Subset**  $\langle P \rangle$  **W** :=  $\{r \mid \text{count} \langle r \rangle = 0\}$

WHILE **W**  $\neq \emptyset$  DO

  Sei  $(A, i) \in \mathbf{W}$  beliebig.

  Entferne  $(A, i)$  aus **W**.

  IF  $A \notin \text{result}$  THEN

**result** := **result**  $\cup \{A\}$

    FORALL  $r \in \text{rhs} \langle A \rangle$  DO

      count  $\langle r \rangle --$

      IF count  $\langle r \rangle = 0$  DO **W** := **W**  $\cup \{r\}$

      ENDIF

    ENDFORALL

  ENDIF

ENDWHILE

Alle prod. Nichtterminale sind nun in **result**.

Laufzeit linear in der Größe der Grammatik.

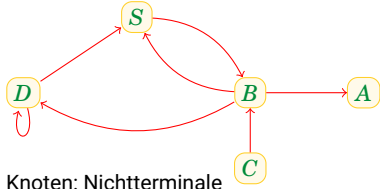
Verwende Tabelle: **Bsp.:**

	$S \rightarrow aBB$	$C \rightarrow a$	$D \rightarrow BD$	$W$	result
$C \rightarrow a$	1	0	2	$(C \rightarrow a)$	$\emptyset$
$B \rightarrow Sd$	0	0	1	$(B \rightarrow C)$	$(C)$
					$(S, A, B, C)$

- Um den Test „ $A \in \text{result}$ “ einfach zu machen, repräsentiert man die Menge **result** durch ein Array.
- $W$  wie auch die Mengen  $\text{rhs}[A]$  werden dagegen als Listen repräsentiert.

Der Algorithmus funktioniert auch, um kleinste Lösungen von Boole'schen Ungleichungssystemen zu bestimmen.

- Leerheitsproblem:**  $\mathcal{L}(G) \neq \emptyset \Leftrightarrow S$  produktiv
- Für Erreichbarkeit: **Abhängigkeitsgraph**. **Bsp.:**



**Knoten:** Nichtterminale

**Kanten:**  $(A, B)$ , falls  $B \rightarrow \alpha_1 A \alpha_2 \in P$

$A$  ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von  $A$  nach  $S$  gibt.

- Erreichbarkeit kann mithilfe von DFS in linearer Zeit berechnet werden.
- Damit ist Berechnung von produktiven & erreichbaren Nichtterminalen linear.

- Def.:** Grammatik  $G$  heißt **reduziert**, wenn alle Nichtterminale von  $G$  produktiv & erreichbar.

- Satz:** Zu jeder kontextfreien Grammatik  $G$  mit  $\mathcal{L}(G) \neq \emptyset$  kann in linearer Zeit eine reduzierte Gr.  $G'$  konstruiert werden mit  $\mathcal{L}(G') = \mathcal{L}(G)$ .

**Konstruktion** der reduzierten Grammatik:

- Berechne die Teilmenge  $N_1 \subseteq N$  aller produktiven Nichtterminale von  $G$ .  
Da  $\mathcal{L}(G) \neq \emptyset$ , ist insbesondere  $S \in N_1$ .
- Entferne unproduktive Regeln:  
 $P_1 := \{A \rightarrow \alpha \in P \mid A \in N_1 \wedge \alpha \in (N_1 \cup T)^*\}$
- Berechne die Teilmenge  $N_2 \subseteq N_1$  aller produktiven & erreichbaren Nichtt. von  $G$ .  
Da  $\mathcal{L}(G) \neq \emptyset$ , ist insbesondere  $S \in N_2$ .
- Entferne unerreichbare Regeln:  
 $P_2 := \{A \rightarrow \alpha \in P \mid A \in N_2 \wedge \alpha \in (N_2 \cup T)^*\}$

Ergebnis:  $G' := (N_2, T, P_2, S)$

### 1.2.3 Grundlagen: Kellerautomaten

- Def.:** **Kellerautomat (PDA, push-down automaton):**  $M = (Q, T, \delta, q_0, F)$  mit:

- $Q$  endl. Menge von Zuständen (bzw. Kellersymbolen),
- $q_0 \in Q$  Anfangszustand,
- $F \subseteq Q$  Menge der Endzustände,
- $\delta \subseteq Q^+ \times (T \cup \{\epsilon\}) \times Q^*$  endl. Menge von Übergängen (gelesener Kellerinhalt, gel. Eingabe, hinzuzufügender Kellerinhalt)

- Def.:** **Konfiguration** des PDA:  $(\gamma, w) \in Q^* \times T^*$   
Ein **Berechnungsschritt** des PDA wird durch die Relation  $\vdash \subseteq (Q^* \times T^*)^2$  beschrieben, wobei  $(\alpha\gamma, xw) \vdash (\alpha\gamma', w)$  für  $(\gamma, x, \gamma') \in \delta$ .  
Relation  $\vdash$  hängt vom PDA ab.  
Reflexiv-transitive Hülle:  $\vdash^*$

- Die von  $M$  akzeptierte Sprache ist:  
 $\mathcal{L}(M) = \{w \in T^* \mid \exists f \in F : (q_0, w) \vdash^* (f, \epsilon)\}$
- PDA  $M$  heißt **deterministisch**, falls  $\forall$  paarw. versch. Übergänge  $(\gamma_1, x, \gamma_2), (\gamma'_1, x', \gamma'_2) \in \delta$  gilt: Ist  $\gamma_1$  Suffig von  $\gamma'_1 \Rightarrow x \neq x' \wedge \gamma_2 \neq \gamma'_2$ .
- Satz:**  $\forall$  kontextfr. Grammatik  $G = (N, T, P, S)$  kann PDA konstr. werden mit  $\mathcal{L}(G) = \mathcal{L}(M)$ .

**Konstruktion 1: Shift-Reduce-Parser**

- Eingabe sukzessive auf Keller schieben.

- Liegt oben auf dem Keller eine vollständige rechte Seite (*Handle*), wird dieses durch die zugehörige linke Seite ersetzt (reduziert).

$M_G^{(1)} = (Q, T, \delta, q_0, \{f\})$  mit

- $Q := T \cup N \cup \{q_0, f\}$
- $\delta := \{(\alpha, x, q\alpha) \mid q \in Q, x \in T\} \cup \{(\alpha\alpha, \epsilon, q\alpha) \mid q \in Q, A \rightarrow \alpha \in P\} \cup \{(q_0S, \epsilon, f)\}$

Eigenschaften:

- Folge der Reduktionen entspricht einer reversen Rechts-Abl. für Eingabe.
- Zur Korrektheit:  
Es gilt  $(q, w) \vdash^* (qA, \epsilon) \Leftrightarrow A \rightarrow^* w$
- $M_G^{(1)}$  i. A. nicht-deterministisch.  
Deterministisches Verfahren: identifiziere Reduktionsstellen  $\rightarrow$  LR-Parsing

**Konstruktion 2: Item-Kellerautomat**

- Rekonstruiere eine Linksableitung.
- Expand. Nichtterminale mithilfe Regel.
- Verifiziere sukzessive, dass die gewählte Regel mit der Eingabe übereinstimmt.  
 $\Rightarrow$  die Zustände sind jetzt *Items*.
- Ein *Item* ist eine Regel mit Punkt:  
 $[A \rightarrow \alpha \bullet \beta], A \rightarrow \alpha \beta \in P$   
Der Punkt gibt an, wie weit die Regel bereits abgearbeitet wurde.

$M_G^{(2)} = (Q, T, \delta, q_0, \{f\})$

- $Q := \{\text{alle Items}\} \cup \{[S' \rightarrow \bullet S], [S' \rightarrow S \bullet]\}$  mit neuem Terminalsymbol  $S'$ .

- $q_0 := [S' \rightarrow \bullet S]$

- $f := [S' \rightarrow S \bullet]$

- Übergänge:

**Expansionen:**

$([A \rightarrow \alpha \bullet B \beta], \epsilon, [A \rightarrow \alpha \bullet B \beta] [B \rightarrow \bullet \gamma])$  für  $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$ ,

**Shifts:**

$([A \rightarrow \alpha \bullet a \beta], a, [A \rightarrow \alpha a \bullet \beta])$  für  $A \rightarrow \alpha a \beta \in P$ ,

**Reduce:**

$([A \rightarrow \alpha \bullet B \beta] [B \rightarrow \bullet \gamma], \epsilon, [A \rightarrow \alpha B \bullet \beta])$  für  $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$ ,

*Items* der Form  $[A \rightarrow \alpha \bullet]$  heißen **vollständig**.

Eigenschaften:

- Expansionen bilden Linksableitung.
- Expansionen nicht-deterministisch.  
Durch Vorausschau deterministisch machen  $\rightarrow$  LL-Parsing.

- Zur Korrektheit:  
Es gilt  $\forall$  Item  $[A \rightarrow \alpha \bullet B \beta]$ :  
 $([A \rightarrow \alpha \bullet B \beta], w) \vdash^* ([A \rightarrow \alpha B \bullet \beta], \epsilon) \Leftrightarrow B \rightarrow^* w$

### 1.2.4 Vorausschau-Mengen

- Def.:** Für Menge  $L \subseteq T^*$  sei  $\text{First}_k(L) := \{u \in L \mid |u| < k\} \cup \{u \in T^k \mid \exists v \in T^* : uv \in L\}$  die Menge der Präfixe der Länge  $k$ .

- $\text{First}_k(\_)$  ist verträglich mit Vereinigung und Konkatenation:

- $\text{First}_k(\emptyset) = \emptyset$

- $\text{First}_k(L_1 \cup L_2) = \text{First}_k(L_1) \cup \text{First}_k(L_2)$

- $\text{First}_k(L_1 \cdot L_2) = \text{First}_k(L_1) \cdot \text{First}_k(L_2)$   
 $= \text{First}_k(L_1) \circ \text{First}_k(L_2)$

- $T^{\leq k} := \bigcup_{0 \leq i \leq k} T^i, \mathbb{D}_k := 2^{T^{\leq k}}$

- $\circ : \mathbb{D}_k \times \mathbb{D}_k \rightarrow \mathbb{D}_k$  distributiv in jedem Arg.:

- $L \circ \emptyset = \emptyset, \emptyset \circ L = \emptyset$

- $L \circ (L_1 \cup L_2) = (L \circ L_1) \cup (L \circ L_2)$

- $(L_1 \cup L_2) \circ L = (L_1 \circ L) \cup (L_2 \circ L)$

- Für  $\alpha \in (N \cup T)^*$  sei

$\text{First}_k(\alpha) := \text{First}_k(\{w \in T^* \mid \alpha \rightarrow^* w\})$

- Für  $k \geq 1$  gilt:  
 $\text{First}_k(x) = \{x\}$  für  $x \in T \cup \{\epsilon\}$   
 $\text{First}(\alpha_1 \alpha_2) = \text{First}_k(\alpha_1) \circ \text{First}_k(\alpha_2)$
- Satz:** Die Mengen  $\text{First}_k(A)$  für  $A \in N$  sind die kleinste Lösung des Ungleichungssystems  $\text{First}_k(A) \supseteq \text{First}_k(X_1) \circ \dots \circ \text{First}_k(X_m)$  für alle  $A \rightarrow X_1 \dots X_m \in P$ .  
Berechnung: Fixpunktiteration (s. ??)

### 1.2.5 Top-down Parsing

- Def.:** Eine reduzierte kontextfr. Gr. heißt LL( $k$ ) (**Left-to-right parsing, Leftmost derivation**, Vorausschau der Länge  $k$ ), falls für verschiedene Regeln  $A \rightarrow \alpha, A \rightarrow \alpha' \in P$  und jede Linksableitung  $S \xrightarrow{*} uA\beta$  mit  $u \in T^*$  gilt:  
 $\text{First}_k(\alpha\beta) \cap \text{First}_k(\alpha'\beta) = \emptyset$ .

**Bsp. KfG2:**  $S \rightarrow \text{if}(E)S \text{ else } S \mid \text{while}(E)S \mid E;$   
 $E \rightarrow \text{id}$

ist LL(1).

**Bsp. KfG3:**  $S \rightarrow \text{if}(E)S \text{ else } S \mid \text{if}(E)S \mid \text{while}(E)S \mid E;$   
 $E \rightarrow \text{id}$

ist nicht LL( $k$ ) für jedes  $k > 0$ , denn: Betrachte Regeln  $(S, 0), (S, 1)$ , d. h.  $\alpha = \text{if}(E)S \text{ else } S$  und  $\alpha' = \text{if}(E)S$ , und die Linksableitung  $S \xrightarrow{*} \text{if}(E)S$ , d. h.  $\beta = \epsilon$ . Es gilt offensichtlich:  
 $\text{First}_k(\alpha\beta) \cap \text{First}_k(\alpha'\beta) \supseteq \{\text{if}\} \neq \emptyset \forall k > 0$ .

- LL( $k$ )-Parser:** sieht Fenster der Länge  $k$  der Eingabe; realisiert i. W. Item-Kellerautomaten, tabelliert nächste zu wählende Regeln.

**Bsp.:** bzgl. KfG2:

Zustände: *Items*

Tabellen:	$\text{if}$	$\text{while}$	$\text{id}$
$[\dots \dots \bullet S \dots]$	0	1	2
$[\dots \dots \bullet E \dots]$	-	-	0

- Menge der möglichen nächsten  $k$  Zeichen ist:  $\text{First}_k(\alpha\beta) = \text{First}_k(\alpha) \circ \text{First}_k(\beta)$ , wobei:  $\alpha$  die rechte Seite der passenden Regel,  $\beta$  ein möglicher rechter Kontext von  $A$ .

$\text{First}_k(\beta)$  dynamisch akkumulieren:

- Ein **erweitertes Item** ist ein Paar  $[A \rightarrow \alpha \bullet \gamma, L]$  für  $A \rightarrow \alpha \gamma \in P, L \subseteq T^{\leq k}$ .  
 $L$  wird als Vorausschau-Menge zum Repräsentieren von  $\text{First}_k(\beta)$  für den rechten Kontext  $\beta$  von  $A$  verwendet.

- Erweiterter Item-Kellerautomat:**

Zustände: erweiterte *Items*

Anfangszustand:  $[S' \rightarrow \bullet S, \{\epsilon\}]$

Endzustand:  $[S' \rightarrow S \bullet, \{\epsilon\}]$

Übergänge:

**Expansionen:**

$([A \rightarrow \alpha \bullet B \beta, L], \epsilon, [A \rightarrow \alpha \bullet B \beta, L] [B \rightarrow \bullet \gamma, \text{First}_k(\beta) \circ L])$  für  $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$ ,

**Shifts:**

$([A \rightarrow \alpha \bullet a \beta, L], a, [A \rightarrow \alpha a \bullet \beta, L])$  für  $A \rightarrow \alpha a \beta \in P$ ,

**Reduce:**

$([A \rightarrow \alpha \bullet B \beta, L] [B \rightarrow \bullet \gamma, L'], \epsilon, [A \rightarrow \alpha B \bullet \beta, L])$  für  $A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$

- Vorausschau-Tabelle:**

$M \left[ [A \rightarrow \alpha \bullet B \beta, L], w \right] := \left\{ i \mid (B, i) = (B \rightarrow \gamma), w \in \text{First}_k(\gamma) \circ \text{First}_k(\beta) \circ L \right\}$

- Satz:** reduzierte kontextfr. Gr.  $G$  ist LL( $k$ )  $\Leftrightarrow \forall$  Eingabewort zu jedem Zeitpunkt in der Berechnung des erw. Item-Kellerautomaten  $|M \left[ [A \rightarrow \alpha \bullet B \beta, L], w \right]| \leq 1$  gilt.

Hierbei ist  $[A \rightarrow \alpha \bullet B \beta, L]$  das aktuelle obere Kellersymbol und  $w$  besteht aus den nächst-

ten  $k$  Zeichen der Eingabe (bzw.  $k' < k$  Zeichen, falls Rest der Eingabe nur noch  $k'$  lang).

- erw. Item-Kellerautomat mit  $k$ -Vorausschau-tabelle erlaubt deterministische Rekonstruktion einer Linksableitung für LL( $k$ )-Grammatik.

Bsp. KfG4:  $S \rightarrow \epsilon \mid aSb$

Vorausschau-Tabelle:	$\epsilon$	$a$	$b$
$[S' \rightarrow \bullet S, \{\epsilon\}]$	0	1	-
$[S \rightarrow a \bullet Sb, \{\epsilon\}]$	-	1	0
$[S \rightarrow a \bullet Sb, \{b\}]$	-	1	0

- Anzahl der Vorausschau-Mengen  $L$  kann sehr groß sein.
- Hängt auszuwählende Regel nicht von den Erweiterungen der Items ab, kann man den Item-Kellerautomat auch ohne Erw. benutzen. Hängt auszuw. Regel nur von der aktuellen Vorausschau  $w$  ab, heißt  $G$  auch **stark LL( $k$ )**.

Bsp.: Die Grammatik aus KfG4 ist anhand der Vorausschau-Tabelle offensichtl. stark LL( $k$ ).

- Def.:  $\text{Follow}_k(A) := \cup \{ \text{First}_k(\beta) \mid S \xrightarrow{*} uA\beta \}$
- Def.: Reduzierte kontextfr. Grammatik  $G$  heißt **stark LL( $k$ )**, falls für je zwei verschiedene Regeln  $A \rightarrow \alpha, A \rightarrow \alpha' \in P$  gilt:

$$\text{First}_k(\alpha) \cap \text{Follow}_k(A) \cap \text{First}_k(\alpha') \cap \text{Follow}_k(A) = \emptyset$$

Bsp.: KfG4 ist tatsächlich stark LL( $k$ ), denn:

$$\text{Follow}_1(S) = \{\epsilon, b\} \Rightarrow$$

$$\text{First}_1(\epsilon) \cap \text{Follow}_1(S) = \{\epsilon\} \cap \{\epsilon, b\} = \{\epsilon, b\}$$

$$\text{First}_1(aSb) \cap \text{Follow}_1(S) = \{a\} \cap \{\epsilon, b\} = \{a\}$$

- Ist  $G$  stark LL( $k$ ), können wir die Vorausschau-Tabelle mit Nichtterminalen (statt erw. Items) indizieren:

$$M[B, w] := \begin{cases} i & (B, i) = (B \rightarrow \gamma) \wedge \\ & w \in \text{First}_k(\gamma) \cap \text{Follow}_k(B) \\ - & \text{keine solche Regel existiert} \end{cases}$$

Bsp.: bzgl. KfG4:

	$\epsilon$	$a$	$b$
$S$	0	1	0

- **Satz:**
  - ▷ Jede stark-LL( $k$ )-Gr. ist auch LL( $k$ ). Gilt i. A. nicht andersherum! Ausnahme:
  - ▷ Jede LL(1)-Gr. ist bereits stark LL(1).
- Zu jeder LL( $k$ )-Grammatik kann eine äquivalente starke LL( $k$ )-Gr. konstruiert werden.

### Berechnung von $\text{Follow}_k(B)$ :

Stelle Ungleichungssystem auf:

$$\text{Follow}_k(S) \supseteq \{\epsilon\} \quad (\text{gilt immer})$$

$$\text{Follow}_k(B) \supseteq$$

$$\text{First}_k(X_1) \cap \text{First}_k(X_m) \cap \text{Follow}_k(A)$$

$$\text{für } A \rightarrow \alpha B X_1 \dots X_m \in P$$

Kleinste Lösung ist  $\text{Follow}_k(B)$ .

- Größe der auftretenden Mengen steigt mit  $k$  rapide. In praktischen Systemen wird darum meist nur der Fall  $k = 1$  implementiert.

## 1.2.6 Schnelle Berechnung von Vorausschau-Mengen

- Im Fall  $k = 1$  lassen sich  $\text{First}_1$  und  $\text{Follow}_1$  besonders effizient berechnen.
- Seien  $L_1, L_2 \subseteq T \cup \{\epsilon\}$  mit  $L_1 \neq \emptyset \neq L_2$ . Dann ist:
$$L_1 \circ L_2 = \begin{cases} L_1 & \epsilon \notin L_1 \\ (L_1 \setminus \{\epsilon\}) \cup L_2 & \text{sonst} \end{cases}$$
- Ist  $G$  reduziert, so sind  $\forall A : \text{First}_1(A) \neq \emptyset$ .
- Def.:  $\text{empty}(X) = \text{true} \Leftrightarrow X \rightarrow^* \epsilon$

- Def.: die  $\epsilon$ -freien **First<sub>1</sub>-Mengen** seien:

$$F_\epsilon(a) := \{a\} \text{ für } a \in T,$$

$$F_\epsilon(A) := \text{First}_1(A) \setminus \{\epsilon\} \text{ für } A \in N.$$

- **Berechnung:** Ungleichungssystem für  $F_\epsilon(A)$ :

$$F_\epsilon(A) \supseteq F_\epsilon(X_j)$$

$$\text{falls } A \rightarrow X_1 \dots X_m \in P, j \in \underline{m},$$

$$\text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1}).$$

Bsp.: bzgl. Grammatik aus KfG1:

$$\text{empty}(E) = \text{empty}(T) = \text{empty}(F) = \text{false}$$

### Ungleichungssystem:

$$\triangleright F_\epsilon(S') \supseteq F_\epsilon(E)$$

$$\triangleright F_\epsilon(E) \supseteq F_\epsilon(E)$$

$$\triangleright F_\epsilon(E) \supseteq F_\epsilon(T)$$

$$\triangleright F_\epsilon(T) \supseteq F_\epsilon(T)$$

$$\triangleright F_\epsilon(T) \supseteq F_\epsilon(F)$$

$$\triangleright F_\epsilon(F) \supseteq \{(name, int)\}$$

- **Ungleichungssystem zu  $\text{Follow}_1(A)$ :**

$$\text{Follow}_1(S) \supseteq \{\epsilon\}$$

$$\text{Follow}_1(B) \supseteq F_\epsilon(X_j)$$

$$\text{falls } A \rightarrow \alpha B X_1 \dots X_m \in P,$$

$$\text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1})$$

$$\text{Follow}_1(B) \supseteq \text{Follow}_1(A)$$

$$\text{falls } A \rightarrow \alpha B X_1 \dots X_m \in P,$$

$$\text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_m)$$

Bsp.: bzgl. Grammatik aus KfG1:

$$\triangleright \text{Follow}_1(S') \supseteq \{\epsilon\}$$

$$\triangleright \text{Follow}_1(E) \supseteq \text{Follow}_1(S')$$

$$\triangleright \text{Follow}_1(E) \supseteq \{+, \cdot\}$$

$$\triangleright \text{Follow}_1(T) \supseteq \{*\}$$

$$\triangleright \text{Follow}_1(T) \supseteq \text{Follow}_1(E)$$

$$\triangleright \text{Follow}_1(F) \supseteq \text{Follow}_1(T)$$

- Die Form dieser Ungleichungssysteme ist:

$$x \supseteq y \text{ bzw. } x \supseteq d$$

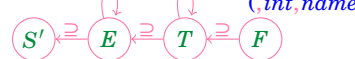
für Variablen  $x, y$  und  $d \in \mathbb{D}$ . Solche Systeme heißen **reine Vereinigungsprobleme** und sind lösbar mit linearem Aufwand.

### Berechnung:

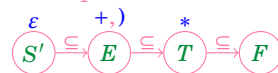
- ▷ Konstruiere den **Variablen-Abhängigkeitsgraph** zum Ungleichungssystem.

Bsp.: bzgl. KfG1 (s. o. für deren Ungl.-S.):

First<sub>1</sub>:



Follow<sub>1</sub>:



- ▷ Innerhalb einer **starken Zusammenhangskomponente** (SZK), d.h. einem maximalen Teilgraph, in dem man von jedem zu jedem Knoten kommen kann, haben alle Variablen den gleichen Wert.

- ▷ Hat eine SZK keine eingehenden Kanten, erhält man ihren Wert, indem man alle Werte innerhalb der SZK vereinigt.

- ▷ Gibt es eingehende Kanten, muss man zusätzlich die Werte an deren Startknoten hinzufügen.

## 1.2.7 Bottom-up-Analyse

- Viele Grammatiken sind nicht LL( $k$ )! Wie parsen wir also diese?
- Def.: Grammatik  $G$  heißt **links-rekursiv**, falls  $A \rightarrow^+ A\beta$  für ein  $A \in N, \beta \in (T \cup N)^*$ .
- Bsp.: KfG1 ist links-rekursiv.
- Satz: Ist  $G$  reduziert & links-rekursiv, dann ist  $\bar{G}$  nicht LL( $k$ )  $\forall k$ .
- $\alpha\gamma$  ist **zuverlässiges Präfix** für das vollst. Item  $[B \rightarrow \gamma \bullet]$ , wenn folgende Berechnung von  $M_G^{(1)}$  existiert:
$$(q_0 \alpha \gamma, v) \vdash (q_0 \alpha B, v) \vdash^* (q_0 S, \epsilon)$$
Dies ist der Fall genau dann, wenn  $S \xrightarrow{*}_R \alpha B v$ .
- Das Item  $[B \rightarrow \gamma \bullet \beta]$  heißt **gültig** für  $\alpha' \Leftarrow S \xrightarrow{*}_R \alpha B v$  mit  $\alpha' = \alpha\gamma$ .
- **Berechnung der Menge zuverlässiger Präfixe** durch NFA, den **charakterist. Automaten**  $c(G)$ :
  - Zustände: Items
  - Anfangszustand:  $[S' \rightarrow \bullet S]$
  - Endzustände:  $\{[B \rightarrow \gamma \bullet] \mid B \rightarrow \gamma \in P\}$
  - Übergänge:

$$1. ([A \rightarrow \alpha \bullet X\beta], X, [A \rightarrow \alpha X \bullet \beta])$$

$$\text{für } X \in (N \cup T), A \rightarrow \alpha X \beta \in P$$

$$2. ([A \rightarrow \alpha \bullet B\beta], \epsilon, [B \rightarrow \bullet \gamma])$$

$$\text{für } A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P$$

- Den **kanonischen LR(0)-Automaten**  $\text{LR}(G)$  erhält man aus  $c(G)$ , indem man  $\epsilon$ -Übergänge entfernt & Teilmengenkonstruktion anwendet.

### Alternative Konstruktion von $\text{LR}(G)$ :

Hilfsfunktion  $\delta_\epsilon^*(q) := q \cup$

$$\{ [B \rightarrow \bullet \gamma] \mid \exists [A \rightarrow \alpha \bullet B' \beta'] \in q,$$

$$\text{Dann, DFA: } \beta \in (N \cup T)^* : B' \rightarrow^* B \beta \}$$

Zustände: Mengen von Items

Anfangszustand:  $\delta_\epsilon^* \{ [S' \rightarrow \bullet S] \}$

Endzustände:  $\{ q \mid \exists A \rightarrow \alpha \in P : [A \rightarrow \alpha \bullet] \in q \}$

Übergänge:  $\delta(q, X) := \delta_\epsilon^* \{ [A \rightarrow \alpha X \bullet \beta] \mid [A \rightarrow \alpha \bullet X \beta] \in q \}$