Compilerbau I

Zusammenfassung wichtiger Elemente der Einführung in den Compilerbau

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.

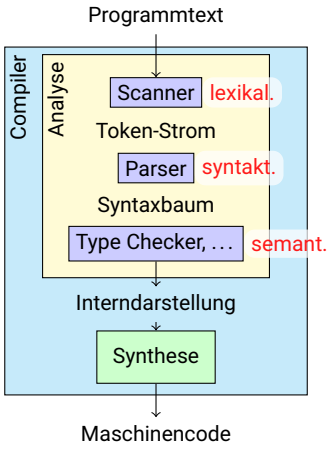
Einzig für Lernzwecke erstellt.

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.

0 Einführung

- **Interpreter**: Keine Vorberechnung auf dem Programmtext erforderlich, Interpreter führt Programmtext „direkt“ aus.
  - ▷ **Bsp.:** Shellskript (`sh`), Javascript (`js`)
  - ▷ Vorteil: geringe Startup-Zeit
  - ▷ Nachteil: Während der Ausführung werden die Bestandteile immer wieder analysiert ⇒ längere Laufzeit.
- **Compiler**: Zwei Phasen:
  1. Übersetzung des Programmtexts in Maschinencode
  2. Ausführung des Maschinencodes
  - ▷ Vorteil: Optimierungen sind im Übersetzungsschritt möglich, z. B. geschicktere Verwaltung der Variablen, d. h. die Ausführung des Programms wird effizienter.
  - ▷ Nachteil: Übersetzung dauert selbst Zeit.

  Lohnt sich bei aufwendigen oder oft ausgeführten Programmen.



1 Analyse-Phase

1.1 Lexikalische Analyse

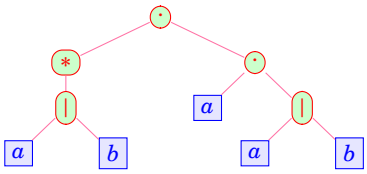
- **Token** ist eine Folge von Zeichen, die zusammen eine Einheit bilden.
- Token-**Klassen**:
  - ▷ Namen (*identifier*) **Bsp.:** `xyz`, `pi`, ...
  - ▷ Konstanten (*constants*) **Bsp.:** `42`, `3.14`, `"abc"`, ...
  - ▷ Operatoren (*operators*) **Bsp.:** `+`, `-`, ...
  - ▷ reservierte Worte (*keywords*) **Bsp.:** `if`, `while`, `int`, ...
- **Sieber**: Vorverarbeitung klassifizierter Tokens
  - ▷ Wegwerfen irrelevanter Teile wie Leerzeichen, Kommentare, ...
  - ▷ Aussondern von **Pragma**s, d. h. Direktiven an den Compiler, die nicht Teil des Programms sind (**Bsp.:** `#include` in C)
  - ▷ Ersetzen der Tokens bestimmter Klassen durch ihre Bedeutung (Konstanten) bzw. Interndarstellung (Namen, die typischerweise zentral in einer **Symbol-Tabelle** verwaltet werden)

  Scanner und Sieber werden i. A. in einer Komponente zusammengefasst, indem man dem Scanner nach Erkennen eines Tokens eine Aktion gestattet.
- Scanner i. A. nicht von Hand programmiert, sondern aus Spezifikation autom. generiert.
  - ▷ Produktivität: Die Komponente lässt sich schneller herstellen.
  - ▷ Korrektheit: Die Komponente realisiert (beweisbar) die Spezifikation.
  - ▷ Effizienz: Der Generator kann die Komponente mit den effizientesten Algorithmen ausstatten.

1.1.1 Grundlagen: Reguläre Ausdrücke

- Programmtext: endl. Alphabet Σ, **Bsp.:** ASCII
- Die Menge der Textabschnitte einer Token-Klasse ist i. A. eine **reguläre Sprache**.
- Reguläre Sprachen kann man mithilfe regulärer Ausdrücke (*reg. expressions*) spezifizieren.
- Def.: Menge $\mathcal{E}_\Sigma$ der (nicht-leeren) **regulären Ausdrücke** ist die kleinste Menge $\mathcal{E}$ mit:
  - ▷ $\varepsilon \in \mathcal{E}$, wobei ε neues Symbol $\notin \Sigma$
  - ▷ $\Sigma \subseteq \mathcal{E}$
  - ▷ $e_1, e_2 \in \mathcal{E} \Rightarrow (e_1 \mid e_2),\ (e_1 \cdot e_2),\ e_1^* \in \mathcal{E}$
- $(, |, \ldots$ nennt man **Meta-Zeichen**.
- Um Klammern zu sparen: $* \succ \cdot \succ |$ und $\cdot$ weglassen
- Reale Spezifikationssprachen bieten zusätzliche Konstrukte wie:
  - ▷ $e? \equiv (\varepsilon \mid e)$ und verzichten auf ε selbst.
  - ▷ $e^+ \equiv (e \cdot e^*)$
  - ▷ $[a-b] \equiv a_i \mid a_{i+1} \mid \ldots \mid a_j$ wenn $a = a_i$, $b = a_j$, $\Sigma = \{a_1 \le a_2 \le \ldots\}$ total geord.
- Def.: Für den regulären Ausdruck $e \in \mathcal{E}_\Sigma$ ist die **spezifizierte Sprache** $[\![e]\!] \subseteq \Sigma^*$ ind. def. durch:
  - ▷ $[\![\varepsilon]\!] = \{\varepsilon\}$
  - ▷ $[\![a]\!] = \{a\}$
  - ▷ $[\![e^*]\!] = ([\![e]\!])^*$, wobei für bel. Sprachen (Mengen) L gilt: $L^* := \{w_1 \cdots w_k \mid k \ge 0,\ w_i \in L\ \forall i \in \underline{k}\}$
  - ▷ $[\![e_1 \mid e_2]\!] = [\![e_1]\!] \cup [\![e_2]\!]$
  - ▷ $[\![e_1 \cdot e_2]\!] = [\![e_1]\!] \cdot [\![e_2]\!]$, wobei für bel. Sprachen (Mengen) L_1, L_2 gilt: $L_1 \cdot L_2 := \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$

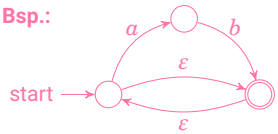
  Bsp.: $[\![a b^* a]\!] = \{a b^n a \mid n \ge 0\}$
- Reguläre Ausdrücke als markierte geordnete Bäume: **Bsp.:** $(a \mid b)^* a (a \mid b)$



Innere Knoten: Operator-Anwendungen, Blätter: einzelne Zeichen oder ε.

1.1.2 Grundlagen: Endliche Automaten

- **Bsp.:**



- Knoten: Zustände, Kanten: Übergänge, Beschriftungen: konsumierter Input
- **ε-NFA** (nicht-deterministischer endl. Automat mit ε-Übergängen): $A = (Q, \Sigma, \delta, I, F)$ mit:
  - ▷ Q endl. Menge von Zuständen
  - ▷ Σ endl. Eingabe-Alphabet
  - ▷ $I \subseteq Q$ Menge der Anfangszustände
  - ▷ $F \subseteq Q$ Menge der akzept. Endzustände
  - ▷ $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ Übergangs-Relation

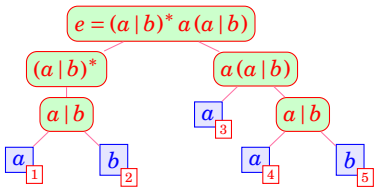
  Andere endl. Automaten:
  - ▷ Gibt es keine ε-Übergänge, ist A ein **NFA**.
  - ▷ Ist $\delta : Q \times \Sigma \to Q$ eine Funktion und $|I| = 1$, heißt A deterministisch (**DFA**).
- **Berechnungen** sind Pfade im Graphen.
- **Akzeptierende Berechnungen** führen von einem Zustand aus I zu einem Zustand aus F.
- Ein **akzeptiertes Wort** ist die Beschriftung einer akzeptierenden Berechnung.

  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** δ^* von δ ist die kleinste Menge δ' mit, für alle $x \in \Sigma \cup \{\varepsilon\}$, $w \in \Sigma^*$:
  - ▷ $(p, \varepsilon, q) \in \delta'$
  - ▷ Wenn $(p, x, p_1) \in \delta' \wedge (p_1, w, q) \in \delta'$ $\Rightarrow (p, xw, q) \in \delta'$

  δ^* 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 ε-NFA konstruiert werden, der die Sprache $[\![e]\!]$ akzeptiert.

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

1. Nummeriere die Blätter. **Bsp.:**



Verwende $\mathrm{num} : \mathcal{E}_\Sigma \to \mathcal{E}_{\mathbb{N}\times\Sigma}$, dazu die Hilfsfkt. $\mathrm{num}' : \mathbb{N} \to (\mathcal{E}_\Sigma \to \mathcal{E}_{\mathbb{N}\times\Sigma})$ definiert für $i \in \mathbb{N}$, $x \in \Sigma$, $e_1, e_2 \in \mathcal{E}_\Sigma$ durch:

- ▷ $\mathrm{num}'(i)(x) := [i, x]$
- ▷ $\mathrm{num}'(i)(\varepsilon) := \varepsilon$
- ▷ $\mathrm{num}'(i)(e_1 \mid e_2) := \mathrm{num}'(i)(e_1) \mid \mathrm{num}'(i + \ell(e_1))(e_2)$
- ▷ $\mathrm{num}'(i)(e_1 \cdot e_2) := \mathrm{num}'(i)(e_1) \cdot \mathrm{num}'(i + \ell(e_1))(e_2)$
- ▷ $\mathrm{num}'(i)(e_1^*) := \mathrm{num}'(i)(e_1)^*$

wobei $\ell : \mathcal{E}_\Sigma \to \mathbb{N}$ die Anzahl Terminalsymbole ist, d. h. für $x \in \Sigma$, $e_1, e_2 \in \mathcal{E}_\Sigma$:

- ▷ $\ell(x) := 1$
- ▷ $\ell(\varepsilon) := 0$
- ▷ $\ell(e_1 \mid e_2) := \ell(e_1) + \ell(e_2)$
- ▷ $\ell(e_1 \cdot e_2) := \ell(e_1) + \ell(e_2)$
- ▷ $\ell(e_1^*) := \ell(e_1)$

Dann ist $\mathrm{num} := \mathrm{num}(1)$.

Bsp.: $\mathrm{num}\big((a \mid b)^* a (a \mid b)\big) = ([1, a] \mid [2, b])^* [3, a] ([4, a] \mid [5, b])$

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

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

   Übergangsrelation:
   - ▷ Für Blätter $r = [i, x]$: $(\bullet r, x, r \bullet) \in \delta$
   - ▷ Für Knoten der Form $r = r_1 \mid r_2$:
     - $(\bullet r,\ \varepsilon,\ \bullet r_1) \in \delta$
     - $(\bullet r,\ \varepsilon,\ \bullet r_2) \in \delta$
     - $(r_1 \bullet,\ \varepsilon,\ r \bullet) \in \delta$
     - $(r_2 \bullet,\ \varepsilon,\ r \bullet) \in \delta$
   - ▷ Für Knoten der Form $r = r_1 \cdot r_2$:
     - $(\bullet r,\ \varepsilon,\ \bullet r_1) \in \delta$
     - $(r_1 \bullet,\ \varepsilon,\ \bullet r_2) \in \delta$
     - $(r_2 \bullet,\ \varepsilon,\ r \bullet) \in \delta$
   - ▷ Für Knoten der Form $r = r_1^*$:
     - $(\bullet r,\ \varepsilon,\ r \bullet) \in \delta$
     - $(\bullet r,\ \varepsilon,\ \bullet r_1) \in \delta$
     - $(r_1 \bullet,\ \varepsilon,\ \bullet r_1) \in \delta$
     - $(r_1 \bullet,\ \varepsilon,\ r \bullet) \in \delta$
   - ▷ Für Knoten der Form $r = r_1?$:
     - $(\bullet r,\ \varepsilon,\ r \bullet) \in \delta$
     - $(\bullet r,\ \varepsilon,\ \bullet r_1) \in \delta$
     - $(r_1 \bullet,\ \varepsilon,\ r \bullet) \in \delta$

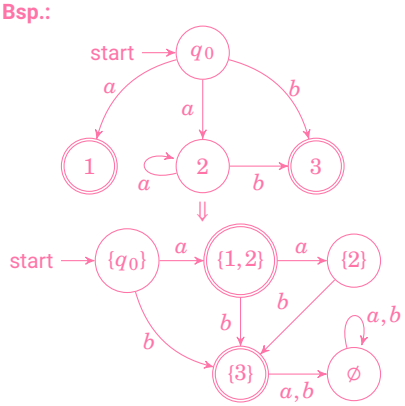
- **Konstruktion** des NFA *ohne* ε-Übergänge:
  Zunächst einige Definitionen:
  - ▷ $\text{empty}[r] = \texttt{true} :\Leftrightarrow \varepsilon \in [\![r]\!]$
    - Für Blätter $r = [i,x]$ ist $\text{empty}[r] := (x \equiv \varepsilon)$
    - $\text{empty}[r_1 \mid r_2] := \text{empty}[r_1] \vee \text{empty}[r_2]$
    - $\text{empty}[r_1 \cdot r_2] := \text{empty}[r_1] \wedge \text{empty}[r_2]$
    - $\text{empty}[r_1^*] := \texttt{true}$
    - $\text{empty}[r_1?] := \texttt{true}$
  - ▷ $\text{first}[r] := \{i \text{ in } r \mid (\bullet r, \varepsilon, \bullet[i,x]) \in \delta^*,\ x \neq \varepsilon\}$
    - Für Blätter $r = [i,x]$ ist $\text{first}[r] := \{i \mid x \neq \varepsilon\}$
    - $\text{first}[r_1 \mid r_2] := \text{first}[r_1] \cup \text{first}[r_2]$
    - $\text{first}[r_1 \cdot r_2] := \begin{cases} \text{first}[r_1] \cup \text{first}[r_2] & \text{empty}[r_1]=\texttt{true} \\ \text{first}[r_1] & \text{empty}[r_1]=\texttt{false} \end{cases}$
    - $\text{first}[r_1^*] := \text{first}[r_1]$
    - $\text{first}[r_1?] := \text{first}[r_1]$
  - ▷ $\text{next}[r] := \{i \in \mathbb{N} \mid (r\bullet, \varepsilon, \bullet[i,x]) \in \delta^*,\ x \neq \varepsilon\}$
    Hier wird induktiv andersherum, also von oben, definiert:
    - Für die Wurzel e ist $\text{next}[e] := \emptyset$.
    - Ist der übergeordnete Knoten $r = \ldots$
      $\ldots r_1 \mid r_2$, dann:
      $\text{next}[r_1] := \text{next}[r]$,
      $\text{next}[r_2] := \text{next}[r]$

      $\ldots r_1 \cdot r_2$, dann:
      $\text{next}[r_1] := \begin{cases} \text{first}[r_2] \cup \text{next}[r] & \text{empty}[r_2=\texttt{true}] \\ \text{first}[r_2] & \text{empty}[r_2=\texttt{false}] \end{cases}$,
      $\text{next}[r_2] := \text{next}[r]$

      $\ldots r_1^*$, dann:
      $\text{next}[r_1] := \text{first}[r_1] \cup \text{next}[r]$

      $\ldots r_1?$, dann:
      $\text{next}[r_1] := \text{next}[r]$
  - ▷ $\text{last}[r] := \{i \text{ in } r \mid ([i,x]\bullet, \varepsilon, r\bullet) \in \delta^*,\ x \neq \varepsilon\}$
    - Für Blätter $r = [i,x]$ ist $\text{last}[r] := \{i \mid x \neq \varepsilon\}$
    - $\text{last}[r_1 \mid r_2] := \text{last}[r_1] \cup \text{last}[r_2]$
    - $\text{last}[r_1 \cdot r_2] := \begin{cases} \text{last}[r_1] \cup \text{last}[r_2] & \text{empty}[r_2]=\texttt{true} \\ \text{last}[r_2] & \text{empty}[r_2]=\texttt{false} \end{cases}$
    - $\text{last}[r_1^*] := \text{last}[r_1]$
    - $\text{last}[r_1?] := \text{last}[r_1]$

  Der **Berry-Sethi-/Glushkow-NFA** A_e ist dann:
  Zustände: $\{q_0\} \uplus \{i \in \mathbb{N} \mid i \text{ in Blatt } [i,x],\ x \neq \varepsilon\}$
  Startzustand: $I := \{q_0\}$
  Endzustände: Falls $\text{empty}[e] = \texttt{false}$, dann $F := \text{last}[e]$; sonst $F := \{q_0\} \uplus \text{last}[e]$.
  (wobei $\uplus$ die disjunkte Vereinigung ist)
  Übergänge: $\forall i, i' \in \mathbb{N}$:
  - ▷ $(q_0, x, i) \in \delta$, falls $i \in \text{first}[e]$ und das Blatt i mit x beschriftet ist.
  - ▷ $(i, y, i') \in \delta$, falls $i' \in \text{next}[[i,x]]$, 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.:



- Viele Tools wie `grep` bauen den DFA nur während der Abarbeitung auf, mit nur genau den Zuständen, wie sie für die jeweilige Eingabe mit Länge n nötig sind. $\Rightarrow \mathcal{O}(n)$
- $\Rightarrow$ Satz: $\forall e \in \mathcal{E}_\Sigma$ kann ein DFA $A = \mathcal{P}(A_e)$ konstruiert werden mit $\mathcal{L}(A) = [\![e]\!]$.

1.1.3 Design eines Scanners

- Eingabe: eine Menge von Regeln:

$$\begin{array}{ll} e_1 & \{\text{action}_1\} \\ e_2 & \{\text{action}_2\} \\ \vdots & \vdots \\ e_k & \{\text{action}_k\} \end{array}$$

  mit $k \in \mathbb{N}$, $e_i \in \mathcal{E}_\Sigma$ und $\varepsilon \notin [\![e_i]\!]\ \forall i \in \underline{k}$.
  Ausgabe: ein Programm, das
  - ▷ von der Eingabe ein *maximales Präfix* w liest, das $e_1 \mid \ldots \mid e_k$ erfüllt,
  - ▷ das minimale i ermittelt mit $w \in [\![e_i]\!]$,
  - ▷ für w dann action_i ausführt.
- **Konstruktion:**
  1. Konstruiere den DFA $\mathcal{P}(A_e)$ (Pot.-A. des Berry-Sethi-NFAs) zu $e := e_1 \mid e_2 \mid \ldots \mid e_k$.
  2. Definiere die Mengen:
     - ▷ $F_1 := \{q \in F \mid q \cap \text{last}[e_1] \neq \emptyset\}$
     - ▷ $F_2 := \{q \in (F \setminus F_1) \mid q \cap \text{last}[e_2] \neq \emptyset\}$
     - ⋮
     - ▷ $F_k := \{q \in (F \setminus (F_1 \cup \cdots \cup F_{k-1})) \mid q \cap \text{last}[e_k] \neq \emptyset\}$

     Für Eingabe w gilt: $\delta^*_\mathcal{P}(q_0, w) \in F_i \Leftrightarrow$ der Scanner für w action_i ausführen soll.
- ***Tokenizing-/Reps' Maximal-Munch-Algorithmus***

```
INPUT: String w ∈ Σ⁺
s := 1; k := 1
FORALL q ∈ Q_P, i ∈ |w| + 1 DO
    fehlversuch[q, i] := false
WHILE true DO
    q := q0; p := ⊥; S := {(q0, k)}
    WHILE k ≤ |w| AND δ(q, w[k]) ≠ ∅ DO
        IF fehlversuch[q, k] = true THEN
            BREAK
        q := δ(q, w[k]); k := k + 1
        IF q ∈ F THEN
            p := q; j := k; S := ∅
        ELSE    S := S ∪ {(q, k)}
    ENDWHILE
    IF p = ⊥ THEN RETURN (failure)
    FORALL (q, k) ∈ S DO
        fehlversuch[q, k] := true
    let i such that p ∈ F_i
    write(w[s, j − 1])
    action_i()
    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:

```
⟨state⟩ {
  e1    {action1    yybegin(state1)}
  e2    {action2    yybegin(state2)}
  ⋮          ⋮              ⋮
  ek    {actionk    yybegin(statek)}
}
```

  Bsp.:

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

1.1.4 Implementierung von DFAs

- Die **Klassifizierungsfunktion** $r : Q \to \mathbb{N}$ ist für $\mathcal{P}(A_e)$ 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: $[q]_r\ \forall q \in Q$
  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. $\overline{Q} := \{r^{-1}(i) \mid i \in \mathbb{N},\ r^{-1}(i) \neq \emptyset\}$
     (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 $\forall \overline{q} \in \overline{Q}\ \forall p_1, p_2 \in \overline{q}\ \forall a \in \Sigma$, ob $r(\delta(p_1,a)) \neq r(\delta(p_2,a))$. In dem Fall muss $\overline{q}$ 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)
     Bsp.:

2					
3					
4					
5					
6					
	1	2	3	4	5

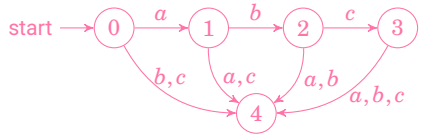
  2. Markiere $\{q, q'\}$, falls:
     - ▷ Erkennungsäq. (GTI): $q \in F,\ q' \notin F$
     - ▷ r-Äquivalenz: $r(q) \neq r(q')$
  3. $\forall$ unmarkierten Paare: teste $\forall a \in \Sigma$, ob $\{\delta(q,a), \delta(q',a)\}$ bereits markiert. Ja $\Rightarrow$ markiere $\{q, q'\}$.
     Bsp.:

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 Zustandspaare, vereinige Übergänge der beiden.

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

Bsp.:



δ	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.:**

δ	0	1	2	3	4
a	1				
b		2			
c			3		

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

δ	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.:**

δ	0	1	2	3	4
a	1		2	1	
b	3	2		0	

$\Downarrow$

δ	0	1	2	3	4	5
a			1		2	1
b	3	2		0		

$\text{displacement}(a) := 2$,
$\text{displacement}(b) = 0$

1.2 Syntaktische Analyse

- Zusammenfassen von Tokens mit **Parser** zu größeren Programmeinheiten, **Bsp.:**
  - ▷ Ausdrücke
  - ▷ Statements
  - ▷ bedingte Verzweigungen (`if`)
  - ▷ Schleifen (`while`, `for`), ...
- 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:** $G = (N, T, P, S)$
  - ▷ N: Menge der Nichtterminale
  - ▷ T: endl. Menge der Terminale
  - ▷ P: Menge der Produktionen/Regeln der Form $A \to \alpha$ mit $A \in N$, $\alpha \in (N \cup T)^*$
  - ▷ $S \in N$ Startsymbol

  Konvention: In Beispielen ist Spezifikation von N und T i. A. implizit anhand ggb. Regeln P.

  Bsp.: $S \to aSb$
  $S \to \varepsilon$

  spezifizierte Sprache: $\{a^n b^n \mid n \geq 0\}$
- Sammle & nummeriere $\forall$ Nichtterminal A die rechten Seiten: (A, j) ist j-te Regel von A.
  Bsp. KfG1: $E \to E + T\ ^0 \mid T\ ^1$
  $T \to T * F\ ^0 \mid F\ ^1$
  $F \to (E)\ ^0 \mid name\ ^1 \mid int\ ^2$
- Grammatiken sind **Wortersetzungssysteme**. Folge v. Ersetzungsschritten heißt **Ableitung**.

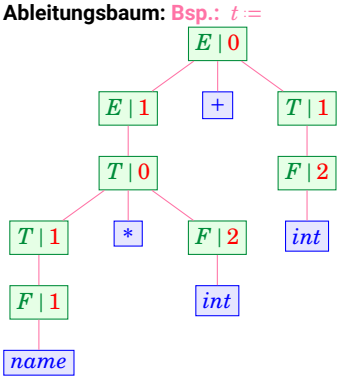
Bsp.: bzgl. Grammatik KfG1:

$$\begin{aligned}
\underline{E} &\to \underline{E} + T && \text{mit Regel } 0\\
&\to \underline{T} + T && \text{mit Regel } 1\\
&\to \underline{T} * \underline{F} + T && \text{mit Regel } 0\\
&\to \underline{T} * int + T && \text{mit Regel } 2\\
&\to \underline{F} * int + T && \text{mit Regel } 1\\
&\to name * int + \underline{T} && \text{mit Regel } 1\\
&\to name * int + \underline{F} && \text{mit Regel } 1\\
&\to name * int + int && \text{mit Regel } 2
\end{aligned}$$

Formal: $\to$ ist Relation auf $(N \cup T)^*$, wobei $\alpha \to \alpha' :\Leftrightarrow \alpha = \alpha_1 A \alpha_2 \wedge \alpha' = \alpha_1 \beta \alpha_2$ für ein $A \to \beta \in P$, $A \in N$, $\alpha, \alpha', \alpha_1, \alpha_2, \beta \in (N \cup T)$.
Reflexiv-transitiver Abschluss von $\to$ sei $\to^*$.

Bemerkungen:
- ▷ $\to$ hängt von Grammatik ab.
- ▷ In jedem Ableitungsschritt kann Stelle & Regel zum Ersetzen ausgesucht werden.
- ▷ Von Grammatik G spezifizierte Sprache ist $\mathcal{L}(G) := \{w \in T^* \mid S \to^* w\}$.

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



innere Knoten: Regel-Anwendungen,
Wurzel: Regel-Anwendung für A,
Blätter: Terminale oder ε

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 linkste bzw. rechteste 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.

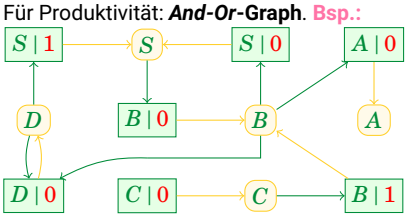
  DFS (*depth-first search*): Tiefensuche
- Konkatenation der Blätter eines Ableitungsbaums t heißt $\text{yield}(t)$.
  Bsp.: $\text{yield}(t) = name * int + int$
- Die Grammatik G heißt **eindeutig**, falls es $\forall w \in T^*$ max. einen Ableitungsbaum t von S gibt mit $\text{yield}(t) = w$.

1.2.2 Überflüssige Nichtterminale & Regeln

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

  Bsp.: $S \to aBB \mid bD$
  $A \to Bc$
  $B \to Sd \mid C$
  $C \to a$
  $D \to BD$

  Produktiv: S, A, B, C; Erreichbar: S, B, C, D
- 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 $(B, i) \equiv B \to \alpha_1 A \alpha_2$

Algorithmus für Produktivität:

Subset $\langle N \rangle$ `result` $:= \emptyset$ (d. h. `result` $\subseteq N$)
int `count`$[P]$
Subset $\langle P \rangle$ `rhs`$[N]$
ForAll $A \in N$ Do `rhs`$[A] := \emptyset$
ForAll $(A, i) \in P$ Do
  `count`$[(A, i)] := 0$
  `init`(A, i)
  ↳ gehe rechte Seite der Produktion (A, i) durch. Für jedes versch. Nichtt. X darin: `count`$[(A, i)]$++ und `rhs`$[X]$.add$((A, i))$.
EndForAll
Subset $\langle P \rangle$ $W := \{r \mid$ `count`$[r] = 0\}$
While $W \neq \emptyset$ Do
  Sei $(A, i) \in W$ beliebig.
  Entferne (A, i) aus W.
  If $A \notin$ `result` Then
    `result` $:=$ `result` $\cup \{A\}$
    ForAll $r \in$ `rhs`$[A]$ Do
      `count`$[r]$ − −
      If `count`$[r] = 0$ Do $W := W \cup \{r\}$
      EndIf
    EndForAll
  EndIf
EndWhile

Alle prod. Nichtterminale sind nun in `result`.

Verwende Tabelle: **Bsp.:**

	$S \to aBB$	⋯	$C \to a$	$D \to BD$	W	`result`
	1	⋯	0	2	$\{C \to a\}$	$\emptyset$
$C \to a$	1	⋯	0	2	$\{B \to C\}$	$\{C\}$
⋮	⋮	⋱	⋮	⋮	⋮	⋮
$B \to Sd$	0	⋯	0	1	$\emptyset$	$\{S, A, B, C\}$

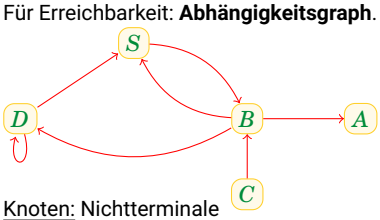
Spalten: Alle Regeln, W, `result`.
Zeilen: nacheinander Regeln aus W
Zellen: `count`

Eigenschaften des Algorithmus:
- ▷ Initialisierung erfordert lineare Laufzeit.
- ▷ Jede Regel wird maximal einmal in W eingefügt.
- ▷ Jede Regel in W hat auf der linken Seite ein produktives Nichtterminal stehen.
- ▷ Jedes A wird maximal einmal in `result` eingefügt.

$\Rightarrow$ Laufzeit linear in der Größe der Grammatik.
- ▷ Um den Test „$A \in$ `result`“ einfach zu machen, repräsentiert man die Menge `result` durch ein Array.
- ▷ W wie auch die Mengen `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 $B \to \alpha_1 A \alpha_2 \in P$
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:

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

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),
  - T Eingabealphabet,
  - $q_0 \in Q$ Anfangszustand,
  - $F \subseteq Q$ Menge der Endzustände,
  - $\delta \subseteq Q^+ \times (T \cup \{\varepsilon\}) \times Q^*$ 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 bezeichnen wir mit $\vdash^*$.
- Die von M akzeptierte Sprache ist:
  $\mathcal{L}(M) = \{w \in T^* \mid \exists f \in F : (q_0, w) \vdash^* (f, \varepsilon)\}$
- PDA M heißt **deterministisch**, falls $\forall$ paarw. versch. Übergänge $(\gamma_1, x, \gamma_2), (\gamma_1', x', \gamma_2') \in \delta$ gilt: Ist γ_1 Suffix von $\gamma_1' \Rightarrow x \neq x' \wedge x \neq \varepsilon \neq x'$.
- 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
- $Q := T \cup N \uplus \{q_0, f\}$
- $\delta := \{(q, x, qx) \mid q \in Q,\ x \in T\} \cup \{(q\alpha, \varepsilon, qA) \mid q \in Q,\ A \to \alpha \in P\} \cup \{(q_0 S, \varepsilon, f)\}$

Eigenschaften:
- Folge der Reduktionen entspricht einer reversen Rechts-Abl. für Eingabe.
- Zur Korrektheit:
  Es gilt $(q, w) \vdash^* (qA, \varepsilon) \Leftrightarrow A \to^* w$
- $M_G^{(1)}$ i. A. nicht-deterministisch.
  Deterministisches Verfahren: identifiziere Reduktionsstellen → **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:
  $[A \to \alpha \bullet \beta], \quad A \to \alpha\beta \in P$
  Der Punkt gibt an, wie weit die Regel bereits abgearbeitet wurde.

$M_G^{(2)} = (Q, T, \delta, q_0, \{f\})$
- $Q :=$ {alle *Items*} $\cup$ $\{[S' \to \bullet S], [S' \to S \bullet]\}$
  mit neuem Terminalsymbol S'.
- $q_0 := [S' \to \bullet S]$
- $f := [S' \to S \bullet]$
- Übergänge:
  Expansionen:
  $([A \to \alpha \bullet B\beta], \varepsilon, [A \to \alpha \bullet B\beta]\,[B \to \bullet\gamma])$
  für $A \to \alpha B\beta,\ B \to \gamma \in P$,
  ***Shifts*:**
  $([A \to \alpha \bullet a\beta], a, [A \to \alpha a \bullet \beta])$
  für $A \to \alpha a \beta \in P$,
  ***Reduce*:**
  $([A \to \alpha \bullet B\beta]\,[B \to \gamma \bullet], \varepsilon, [A \to \alpha B \bullet \beta])$
  für $A \to \alpha B\beta,\ B \to \gamma \in P$,

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

Eigenschaften:
- Expansionen bilden Linksableitung.
- Expansionen nicht-deterministisch.
  Durch Vorausschau deterministisch machen → **LL-Parsing**.
- Zur Korrektheit:
  Es gilt $\forall$ *Item* $[A \to \alpha \bullet B\beta]$:
  $([A \to \alpha \bullet B\beta], w) \vdash^* ([A \to \alpha B \bullet \beta], \varepsilon) \Leftrightarrow B \to^* w$

1.2.4 Vorausschau-Mengen

- Def.: Für Menge $L \subseteq T^*$ sei $\mathrm{First}_k(L) := \{u \in L \mid |u| < k\} \cup \{u \in T^k \mid \exists v \in T^* : uv \in L\}$ die **Menge der Präfixe** der Länge k.
- $\mathrm{First}_k(\_)$ ist verträglich mit Vereinigung und Konkatenation:
  - $\mathrm{First}_k(\emptyset) = \emptyset$
  - $\mathrm{First}_k(L_1 \cup L_2) = \mathrm{First}_k(L_1) \cup \mathrm{First}_k(L_2)$
  - $\mathrm{First}_k(L_1 \cdot L_2) = \mathrm{First}_k(\mathrm{First}_k(L_1) \cdot \mathrm{First}_k(L_2)) =: \mathrm{First}_k(L_1) \odot \mathrm{First}_k(L_2)$
- $T^{\leq k} := \bigcup_{0 \leq i \leq k} T^i, \quad \mathbb{D}_k := 2^{T^{\leq k}}$
- $\odot : \mathbb{D}_k \times \mathbb{D}_k \to \mathbb{D}_k$ distributiv in jedem Arg.:
  - $L \odot \emptyset = \emptyset, \quad \emptyset \odot L = \emptyset$
  - $L \odot (L_1 \cup L_2) = (L \odot L_1) \cup (L \odot L_2)$
  - $(L_1 \cup L_2) \odot L = (L_1 \odot L) \cup (L_2 \odot L)$
- Für $\alpha \in (N \cup T)^*$ sei $\mathrm{First}_k(\alpha) := \mathrm{First}_k(\{w \in T^* \mid \alpha \to^* w\})$
- Für $k \geq 1$ gilt:
  - $\mathrm{First}_k(x) = \{x\}$ für $x \in T \cup \{\varepsilon\}$
  - $\mathrm{First}(\alpha_1\alpha_2) = \mathrm{First}_k(\alpha_1) \odot \mathrm{First}_k(\alpha_2)$
- Satz: Die Mengen $\mathrm{First}_k(A)$ für $A \in N$ sind die kleinste Lösung des Ungleichungssystems $\mathrm{First}_k(A) \supseteq \mathrm{First}_k(X_1) \odot \cdots \odot \mathrm{First}_k(X_m)$ für alle $A \to X_1 \ldots X_m \in P$.

  Bsp.: bzgl. Grammatik KfG1:
  - $\mathrm{First}_2(E) \supseteq \mathrm{First}_2(E{+}T) \supseteq \mathrm{First}_2(E) \odot \{+\} \odot \mathrm{First}_2(T)$
  - $\mathrm{First}_2(T) \supseteq \mathrm{First}_2(T{*}F) \supseteq \mathrm{First}_2(T) \odot \{*\} \odot \mathrm{First}_2(F)$
  - $\mathrm{First}_2(F) \supseteq \mathrm{First}_2((E)) \supseteq \{(\} \odot \mathrm{First}_2(E) \odot \{)\}$
  - $\mathrm{First}_2(E) \supseteq \mathrm{First}_2(T)$
  - $\mathrm{First}_2(T) \supseteq \mathrm{First}_2(F)$
  - $\mathrm{First}_2(F) \supseteq \{name, int\}$

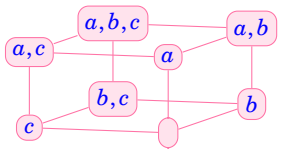
  Berechnung: Fixpunktiteration (s. 1.2.5), denn:
  - Die Menge $\mathbb{D}_k$ der möglichen Werte für $\mathrm{First}_k(A)$ ist vollständiger Verband.
  - Operatoren auf den rechten Seiten sind monoton, d. h. hier: verträglich mit $\subseteq$.

1.2.5 Exkurs: Vollständige Verbände

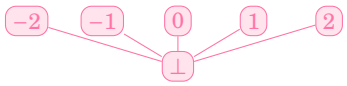
Dieses Kapitel ist nicht klausurrelevant in SS'20.

- Def.: Eine Menge $\mathbb{D}$ mit Relation $\sqsubseteq \subseteq \mathbb{D} \times \mathbb{D}$ ist ein **Verband** (***lattice***), falls $\forall a, b, c \in \mathbb{D}$:
  - $a \sqsubseteq a$ (reflexiv)
  - $a \sqsubseteq b \wedge b \sqsubseteq a \Rightarrow a = b$ (antisymmetrisch)
  - $a \sqsubseteq b \wedge b \sqsubseteq c \Rightarrow a \sqsubseteq c$ (transitiv)

Bsp.:
- $\mathbb{D} := 2^{\{a,b,c\}}$ mit Relation $\subseteq$



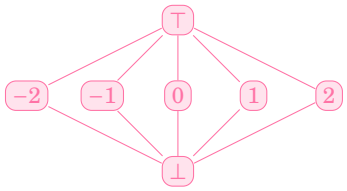
- $\mathbb{Z}$ mit Relation $=$, $\mathbb{Z}$ mit Relation $\leq$
- $\mathbb{Z}_\perp := \mathbb{Z} \cup \{\perp\}$ mit der Ordnung:



- Def.: $d \in \mathbb{D}$ heißt **obere Schranke** für $X \subseteq \mathbb{D}$, falls $x \sqsubseteq d \ \forall x \in X$.
- Def.: d heißt **kleinste obere Schranke** (***lowest upper bound*, *lub***) von $X \subseteq \mathbb{D}$, falls:
  - d eine obere Schranke für X ist
  - $d \sqsubseteq y$ für jede obere Schranke y für X

  Schreibe $d =: \sup X =: \bigsqcup X$.
- Def.: Ein **vollständiger Verband** (***complete lattice*, *cl***) $\mathbb{D}$ ist eine Halbordnung $\sqsubseteq$, in der jede Teilmenge $X \subseteq \mathbb{D}$ ein *lub* $\bigsqcup X \in \mathbb{D}$ besitzt.

  Bsp.:
  - $\mathbb{D} := 2^{\{a,b,c\}}$ ist ein *cl*.
  - $\mathbb{Z}$ mit $=$ oder $\leq$ sowie $\mathbb{Z}_\perp$ sind kein *cl*!
  - $\mathbb{Z}_\perp^\top := \mathbb{Z} \cup \{\perp, \top\}$ ist der **flache Verband**:



- Satz: In jedem *cl* $\mathbb{D}$ besitzt jedes $X \subseteq \mathbb{D}$ eine **größte untere Schranke** $\bigsqcap X := \inf X \in \mathbb{D}$.
  Und zwar gilt: $\bigsqcap X = \bigsqcup U_X$,
  wobei $U_X := \{u \in \mathbb{D} \mid \forall x \in X : u \sqsubseteq x\} \subseteq \mathbb{D}$ die Menge der unteren Schranken von X.
- Def.: Abbildung $f : \mathbb{D}_1 \to \mathbb{D}_2$ heißt **monoton**, falls $\forall a, b \in \mathbb{D}_1 : a \sqsubseteq_1 b \Rightarrow f(a) \sqsubseteq_2 f(b)$.

  Bsp.: $\mathbb{D}_1 := \mathbb{D}_2 := \mathbb{Z}$ mit Ordnung $\leq$. Dann sind $\mathrm{inc}\,x := x + 1$ und $\mathrm{dec}\,x := x - 1$ monoton; $\mathrm{inv}\,x := -x$ ist *nicht* monoton.
- **Ungleichungssystem über Verbänden:**

  GESUCHT: möglichst kleine Lösung für $x_i \sqsupseteq f_i(x_1, \ldots, x_n),\ i \in \underline{n} := \{1, 2, \ldots, n\}$, wobei $f_i : \mathbb{D}^n \to \mathbb{D}$ monoton $\forall i \in \underline{n}$.

  Lösung: **Fixpunktiteration**

  Sei $F : \mathbb{D}^n \to \mathbb{D}^n,\ F(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ mit $y_i := f_i(x_1, \ldots, x_n)\ \forall i \in \underline{n}$.
  Sind alle f_i monoton, so ist auch F monoton.
  Konstruiere $\perp, F\perp, F^2\perp, F^3\perp, \ldots$
  Gilt $F^k\perp = F^{k+1}\perp$, so ist eine Lösung gefunden. Ist $\mathbb{D}$ endlich, so gibt es k immer.

  Bsp. FI1: $\mathbb{D} := 2^{\{a,b,c\}},\ \sqsubseteq := \subseteq$
  $x_1 \supseteq \{a\} \cup x_3, \quad x_2 \supseteq x_3 \cap \{a, b\}, \quad x_3 \supseteq x_1 \cup \{c\}$
  Iteration:

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$	$\{a,c\}$	$\{a,c\}$	$\{a,c\}$
x_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$	$\{a\}$
x_3	$\emptyset$	$\{c\}$	$\{a,c\}$	$\{a,c\}$	$\{a,c\}$

  Noch zu klären: *kleinste* Lösung?
- Def.: Eine Funktion $f : \mathbb{D} \to \mathbb{D}$ heißt **stetig**, falls $\forall$ aufsteigende Kette $d_0 \sqsubseteq \cdots \sqsubseteq d_m \sqsubseteq \ldots$ gilt: $f(\bigsqcup_{m \geq 0} d_m) = \bigsqcup_{m \geq 0} (f(d_m))$.
- Jede stetige Funktion ist auch monoton.
- Def.: Eine Halbordnung $(\mathbb{D}, \sqsubseteq)$ heißt **vollständig** (***complete partial order*, *CPO***), falls alle aufsteigenden Ketten ein *lub* haben.
- Jeder vollständige Verband ist eine *CPO*.
- Satz (**Kleene**): In einer vollständigen Halbordnung $\mathbb{D}$ hat jede stetige Funktion $f : \mathbb{D} \to \mathbb{D}$

einen kleinsten Fixpunkt $d_0 = \bigsqcup_{k\geq 0} f^k \bot$.

- Für $P := \{x \in \mathbb{D} \mid x \sqsupseteq f(x)\}$ ist der kleinste Fixpunkt $d_0 = \min P$ (d. h. untere Schr. & in P).
- Die **kleinste Lösung eines Ungleichungssystems** ist der kleinste Fixpunkt von F.
- ***Round-Robin*-Iteration (*RR*):**

  Benutze bei der Iteration nicht die Werte der letzten Iteration, sondern die jeweils aktuellen.

  Bsp.: Bezüglich Ungleichungssystem aus FI1:

	0	1	2	3
x_1	$\emptyset$	$\{a\}$	$\{a,c\}$	$\{a,c\}$
x_2	$\emptyset$	$\emptyset$	$\{a\}$	$\{a\}$
x_3	$\emptyset$	$\{a,c\}$	$\{a,c\}$	$\{a,c\}$

Algorithmus *Round-Robin*-Iteration:

```
FORALL i ∈ n DO x_i := ⊥ ENDFORALL
REPEAT
  finished := true
  FORALL i ∈ n DO
    new := f_i(x_1, ..., x_n)
    IF ¬(x_i ⊒ new)
      finished := false
      x_i := x_i ⊔ new
    ENDIF
  ENDFORALL
UNTIL finished
```

- Länge h der längsten echt aufsteigenden Kette nennen wir **Höhe** von $\mathbb{D}$.

  Bsp.: Für First_k ist die Höhe des Verbands exponentiell in k.
  $\Rightarrow$ man beschränkt sich i. A. auf kleine k.
- Anzahl Runden in *RR*-Iteration ist beschränkt durch $\mathcal{O}(n \cdot h)$, wobei n die Anzahl Variablen.
- Effizienz von *RR*-Iteration hängt von Anordnung der Variablen ab.

1.2.6 *Top-down Parsing*

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

  Bsp. KfG2: $S \to \mathit{if}\ (E)\ S\ \mathit{else}\ S \mid \mathit{while}\ (E)\ S \mid E;$
  $E \to \mathit{id}$

  ist $\text{LL}(1)$.

  Bsp. KfG3: $S \to \mathit{if}\ (E)\ S\ \mathit{else}\ S \mid \mathit{if}\ (E)\ S \mid \mathit{while}\ (E)\ S \mid E;$
  $E \to \mathit{id}$

  ist nicht $\text{LL}(k)$ für jedes $k > 0$, denn: Betrachte Regeln $(S,0),(S,1)$, d. h. $\alpha = \mathit{if}\ (E)\ S\ \mathit{else}\ S$ und $\alpha' = \mathit{if}\ (E)\ S$, und die Linksableitung $S \to_L^* \mathit{if}\ (\mathit{id})\ S$, d. h. $\beta = \varepsilon$. Es gilt offensichtl.: $\text{First}_k(\alpha\beta) \cap \text{First}_k(\alpha'\beta) \supseteq \{\mathit{if}\} \neq \emptyset\ \forall k > 0$.
- **$\text{LL}(k)$-Parser:** sieht Fenster der Länge k der Eingabe; realisiert i. W. *Item*-Kellerautomaten, tabelliert nächste zu wählende Regeln.

  Bsp.: bzgl. KfG2:

  Zustände: *Items*

Tabelle:	*if*	*while*	*id*
$[\ldots \to \ldots \bullet S \ldots]$	0	1	2
$[\ldots \to \ldots \bullet E \ldots]$	–	–	0

- Menge der möglichen nächsten k Zeichen ist: $\text{First}_k(\alpha\beta) = \text{First}_k(\alpha) \odot \text{First}_k(\beta)$, wobei: α die rechte Seite der passenden Regel, β ein möglicher rechter Kontext von A.
  $\text{First}_k(\beta)$ dynamisch akkumulieren:
- Ein **erweitertes *Item*** ist ein Paar $[A \to \alpha \bullet \gamma, L]$ für $A \to \alpha\gamma \in P$, $L \subseteq T^{\leq k}$.
  L wird als Vorausschau-Menge zum Repräsentieren von $\text{First}_k(\beta)$ für den rechten Kontext β von A verwendet.
- **Erweiterter *Item*-Kellerautomat:**

  Zustände: erweiterte *Items*

  Anfangszustand: $[S' \to \bullet S, \{\varepsilon\}]$

  Endzustand: $[S' \to S \bullet, \{\varepsilon\}]$

  Übergänge:

  Expansionen:
  $([A \to \alpha \bullet B\beta, L], \varepsilon,$
  $[A \to \alpha \bullet B\beta, L]\,[B \to \bullet\gamma, \text{First}_k(\beta) \odot L])$
  für $A \to \alpha B\beta$, $B \to \gamma \in P$,

  ***Shifts*:**
  $([A \to \alpha \bullet a\beta, L], a, [A \to \alpha a \bullet \beta, L])$
  für $A \to \alpha a\beta \in P$,

  ***Reduce*:**
  $([A \to \alpha \bullet B\beta, L]\,[B \to \gamma \bullet, L'], \varepsilon,$
  $[A \to \alpha B \bullet \beta, L])$ für $A \to \alpha B\beta$, $B \to \gamma \in P$
- **Vorausschau-Tabelle:**
  $$M\left[[A \to \alpha \bullet B\beta, L], w\right] := \left\{ i \;\middle|\; (B,i) = (B \to \gamma),\ w \in \text{First}_k(\gamma) \odot \text{First}_k(\beta) \odot L \right\}$$
- Satz: reduzierte kontextfr. Gr. G ist $\text{LL}(k)$ $\Leftrightarrow \forall$ Eingabewort zu jedem Zeitpunkt in der Berechnung des erw. *Item*-Kellerautomaten $\left|M\left[[A \to \alpha \bullet B\beta, L], w\right]\right| \leq 1$ gilt.

  Hierbei ist $[A \to \alpha \bullet B\beta, L]$ das aktuelle oberste Kellersymbol und w besteht aus den nächsten k Zeichen der Eingabe (bzw. $k' < k$ Zeichen, falls Rest der Eingabe nur noch k' lang).
- erw. *Item*-Kellerautomat mit k-Vorausschautabelle erlaubt deterministische Rekonstruktion einer Linksableitung für $\text{LL}(k)$-Grammatik.

  Bsp. KfG4: $S \to \varepsilon \mid aSb$

Vorausschau-Tabelle:	ε	a	b
$[S' \to \bullet S, \{\varepsilon\}]$	0	1	–
$[S \to a \bullet Sb, \{\varepsilon\}]$	–	1	0
$[S \to a \bullet Sb, \{b\}]$	–	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 $\text{LL}(k)$**.

  Bsp.: Die Grammatik aus KfG4 ist anhand der Vorausschau-Tabelle offensichtl. stark $\text{LL}(k)$.
- Def.: $\text{Follow}_k(A) := \bigcup \{\text{First}_k(\beta) \mid S \to_L^* uA\beta\}$
- Def.: Reduzierte kontextfr. Grammatik G heißt **stark $\text{LL}(k)$**, falls für je zwei verschiedene Regeln $A \to \alpha, A \to \alpha' \in P$ gilt:
  $\text{First}_k(\alpha) \odot \text{Follow}_k(A) \cap \text{First}_k(\alpha') \odot \text{Follow}_k(A) = \emptyset$

  Bsp.: KfG4 ist tatsächlich stark $\text{LL}(k)$, denn:
  $\text{Follow}_1(S) = \{\varepsilon, b\} \Rightarrow$
  $\text{First}_1(\varepsilon) \odot \text{Follow}_1(S) = \{\varepsilon\} \odot \{\varepsilon, b\} = \{\varepsilon, b\}$
  $\text{First}_1(aSb) \odot \text{Follow}_1(S) = \{a\} \odot \{\varepsilon, b\} = \{a\}$
- Ist G stark $\text{LL}(k)$, können wir die Vorausschau-Tabelle mit Nichtterminalen (statt erw. *Items*) indizieren:
  $$M[B,w] := \begin{cases} i & (B,i) = (B \to \gamma) \wedge \\ & w \in \text{First}_k(\gamma) \odot \text{Follow}_k(B) \\ - & \text{keine solche Regel existiert} \end{cases}$$

  Bsp.: bzgl. KfG4:

	ε	a	b
S	0	1	0

- Satz:
  - ▷ Jede stark-$\text{LL}(k)$-Gr. ist auch $\text{LL}(k)$.
    Gilt i. A. nicht andersherum! Ausnahme:
  - ▷ Jede $\text{LL}(1)$-Gr. ist bereits stark $\text{LL}(1)$.
- Zu jeder $\text{LL}(k)$-Grammatik kann eine äquivalente starke $\text{LL}(k)$-Gr. konstruiert werden.
- **Berechnung von $\text{Follow}_k(B)$:**

  Stelle Ungleichungssystem auf:
  $\text{Follow}_k(S) \supseteq \{\varepsilon\}$ (gilt immer)
  $\text{Follow}_k(B) \supseteq \text{First}_k(X_1) \odot \text{First}_k(X_m) \odot \text{Follow}_k(A)$
  für $A \to \alpha B X_1 \ldots X_m \in P$

  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.

1.2.7 Schnelle Berechnung von Vorausschau-Mengen

- Im Fall $k = 1$ lassen sich First_1 und Follow_1 besonders effizient berechnen.
- Seien $L_1, L_2 \subseteq T \cup \{\varepsilon\}$ mit $L_1 \neq \emptyset \neq L_2$. Dann ist:
  $$L_1 \odot L_2 = \begin{cases} L_1 & \varepsilon \notin L_1 \\ (L_1 \setminus \{\varepsilon\}) \cup L_2 & \text{sonst} \end{cases}$$
- Ist G reduziert, so sind $\forall A : \text{First}_1(A) \neq \emptyset$.
- Def.: $\text{empty}(X) = \texttt{true} :\Leftrightarrow X \to^* \varepsilon$
- Def.: die **ε-freien First_1-Mengen** seien:
  $\text{F}_\varepsilon(a) := \{a\}$ für $a \in T$,
  $\text{F}_\varepsilon(A) := \text{First}_1(A) \setminus \{\varepsilon\}$ für $A \in N$.
- **Berechnung:** Ungleichungssystem für $\text{F}_\varepsilon(A)$:

  $\text{F}_\varepsilon(A) \supseteq \text{F}_\varepsilon(X_j)$
  falls $A \to X_1 \ldots X_m \in P$, $j \in \underline{m}$,
  $\text{empty}(X_1) \wedge \cdots \wedge \text{empty}(X_{j-1})$.

  Bsp.: bzgl. Grammatik aus KfG1:
  $\text{empty}(E) = \text{empty}(T) = \text{empty}(F) = \texttt{false}$
  Ungleichungssystem:
  - ▷ $\text{F}_\varepsilon(S') \supseteq \text{F}_\varepsilon(E)$
  - ▷ $\text{F}_\varepsilon(E) \supseteq \text{F}_\varepsilon(E)$
  - ▷ $\text{F}_\varepsilon(E) \supseteq \text{F}_\varepsilon(T)$
  - ▷ $\text{F}_\varepsilon(T) \supseteq \text{F}_\varepsilon(T)$
  - ▷ $\text{F}_\varepsilon(T) \supseteq \text{F}_\varepsilon(F)$
  - ▷ $\text{F}_\varepsilon(F) \supseteq \{(, \mathit{name}, \mathit{int}\}$
- **Ungleichungssystem zu $\text{Follow}_1(A)$:**

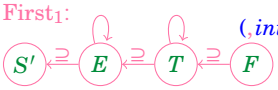
  $\text{Follow}_1(S) \supseteq \{\varepsilon\}$
  $\text{Follow}_1(B) \supseteq \text{F}_\varepsilon(X_j)$
  falls $A \to \alpha B X_1 \ldots X_m \in P$,
  $\text{empty}(X_1) \wedge \cdots \wedge \text{empty}(X_{j-1})$
  $\text{Follow}_1(B) \supseteq \text{Follow}_1(A)$
  falls $A \to \alpha B X_1 \ldots X_m \in P$,
  $\text{empty}(X_1) \wedge \cdots \wedge \text{empty}(X_m)$

  Bsp.: bzgl. Grammatik aus KfG1:
  - ▷ $\text{Follow}_1(S') \supseteq \{\varepsilon\}$
  - ▷ $\text{Follow}_1(E) \supseteq \text{Follow}_1(S')$
  - ▷ $\text{Follow}_1(E) \supseteq \{+,)\}$
  - ▷ $\text{Follow}_1(T) \supseteq \{*\}$
  - ▷ $\text{Follow}_1(T) \supseteq \text{Follow}_1(E)$
  - ▷ $\text{Follow}_1(F) \supseteq \text{Follow}_1(T)$
- Die Form dieser Ungleichungssysteme ist:
  $x \sqsupseteq y$ bzw. $x \sqsupseteq d$
  für Variablen x, y und $d \in \mathbb{D}$. Solche Systeme heißen **reine Vereinigungsprobleme** und sind lösbar mit linearem Aufwand.

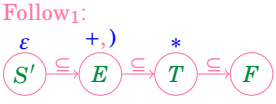
  Berechnung:
  - ▷ Konstruiere den **Variablen-Abhängigkeitsgraph** zum Ungleichungssystem.

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

    First_1:

    Follow_1:

  - ▷ 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 Startkno-

ten hinzufügen.

1.2.8 *Bottom-up*-Analyse

- Viele Grammatiken sind nicht $\mathrm{LL}(k)$!
  Wie parsen wir also diese?
- Def.: Grammatik G heißt **links-rekursiv**, falls $A \to^+ A\beta$ für ein $A \in N$, $\beta \in (T \cup N)^*$.
  Bsp.: KfG1 ist links-rekursiv.
- Satz: Ist G reduziert & links-rekursiv, dann ist G nicht $\mathrm{LL}(k)$ $\forall k$.
- $\alpha\gamma$ ist **zuverlässiges Präfix** für das vollst. *Item* $[B \to \gamma\bullet]$, wenn folgende Berechnung von $M_G^{(1)}$ existiert:
  $(q_0\alpha\gamma, v) \vdash (q_0\alpha B, v) \vdash^* (q_0 S, \varepsilon)$
  Dies ist der Fall genau dann, wenn $S \to_R^* \alpha B v$.
- Das *Item* $[B \to \gamma \bullet \beta]$ heißt **gültig** für $\alpha' \Leftrightarrow S \to_R^* \alpha B v$ mit $\alpha' = \alpha\gamma$.
- **Berechnung der Menge zuverlässiger Präfixe** durch NFA, den **charakterist. Automat** $c(G)$:
  Zustände: *Items*
  Anfangszustand: $[S' \to \bullet S]$
  Endzustände: $\{[B \to \gamma\bullet] \mid B \to \gamma \in P\}$
  Übergänge:
  1. $([A \to \alpha \bullet X\beta], X, [A \to \alpha X \bullet \beta])$
     für $X \in (N \cup T)$, $A \to \alpha X\beta \in P$
  2. $([A \to \alpha \bullet B\beta], \varepsilon, [B \to \bullet\gamma])$
     für $A \to \alpha B\beta, B \to \gamma \in P$
- Den **kanonischen $\mathrm{LR}(0)$-Automaten** $\mathrm{LR}(G)$ erhält man aus $c(G)$, indem man ε-Übergänge entfernt & Teilmengenkonstruktion anwendet.
  Alternative Konstruktion von $\mathrm{LR}(G)$:
  Hilfsfunktion $\delta_\varepsilon^*(q) := q \cup \left\{[B \to \bullet\gamma] \mid \exists [A \to \alpha \bullet B'\beta'] \in q, \beta \in (N \cup T)^* : B' \to^* B\beta\right\}$
  Dann, DFA:
  Zustände: Mengen von *Items*
  Anfangszustand: $\delta_\varepsilon^*\{[S' \to \bullet S]\}$
  Endzustände: $\{q \mid \exists A \to \alpha \in P : [A \to \alpha\bullet] \in q\}$
  Übergänge: $\delta(q, X) := \delta_\varepsilon^*\left\{[A \to \alpha X \bullet \beta] \mid [A \to \alpha \bullet X\beta] \in q\right\}$

↓ Alles folgende nicht mehr klausurrelevant in SS'20.

- **$\mathrm{LR}(0)$-Parser:** Sei $\mathrm{LR}(G) = (Q, T, \delta, q_0, F)$. Dann ist der $\mathrm{LR}(0)$-Parser:
  Zustände: $Q \uplus \{f\}$ (wobei f neu)
  Anfangszustand: q_0 Endzustand: f
  Übergänge:
  Shift: (q, a, pq), falls $q = \delta(p, a) \neq \emptyset$
  Reduce: $(pq_1 \ldots q_m, \varepsilon, pq)$,
  falls $[A \to X_1 \ldots X_m \bullet] \in q_m$, $q = \delta(p, A)$
  Finish: $(q_0 p, \varepsilon, f)$, falls $[S' \to S\bullet] \in p$
  - ▷ Die akzeptierenden Berechnungen des $\mathrm{LR}(0)$-Parsers stehen in 1:1-Beziehung zu denen des *Shift-Reduce*-Parsers $M_G^{(1)}$.
  - ▷ Akzeptierte Sprache ist $\mathcal{L}(G)$.
  - ▷ Die Folge der Reduktionen einer akzept. Berechnung für ein Wort $w \in T$ liefert eine reverse Rechts-Abl. von G für w.
  - ▷ Achtung: Nur deterministisch, wenn jeder Endzustand des kanon. $\mathrm{LR}(0)$-Automaten $\mathrm{LR}(G)$ keine Konflikte enthält. Sonst i. A. nicht-deterministisch!
- 2 Gründe für Nicht-Determinismus v. $\mathrm{LR}(0)$-P.:
  - ▷ ***Reduce-Reduce*-Konflikt:**
    $[A \to \gamma\bullet], [A' \to \gamma'\bullet] \in q$
    mit $A \neq A' \vee \gamma \neq \gamma'$.
  - ▷ ***Shift-Reduce*-Konflikt:**
    $[A \to \gamma\bullet], [A' \to \alpha \bullet a\beta] \in q$
    mit $a \in T$, $q \in Q$.

  Solche Zustände q heißen **ungeeignet**.
- Satz: Die reduzierte Grammatik G ist $\mathrm{LR}(0) \Leftrightarrow$ der kanonische $\mathrm{LR}(0)$-Automat $\mathrm{LR}(G)$ keine ungeeigneten Zustände enthält.
- Def.: Die reduzierte kontextfreie Grammatik G heißt **$\mathrm{LR}(k)$-Grammatik**, falls für Wörter $w, x \in T^*$ mit $\mathrm{First}_k(w) = \mathrm{First}_k(x)$ aus
  $S \to_R^* \alpha A w \to \alpha\beta w$ und
  $S \to_R^* \alpha' A' w' \to \alpha\beta x$
  folgt: $\alpha = \alpha' \wedge A = A' \wedge w' = x$
- Ein **$\mathrm{LR}(k)$-*Item*** ist ein Paar $[B \to \alpha \bullet \beta, x]$ mit $x \in \mathrm{Follow}_k(B)$.
  Dieses *Item* ist gültig für $\gamma\alpha$, falls $S \to_R^* \gamma B w$ mit $\{x\} = \mathrm{First}_k(w)$.
- **Berechnung** der gült. $\mathrm{LR}(k)$-*Items* für zuverl. Präf.: **charakterist. $\mathrm{LR}(k)$-Automat** $c(G, k)$:
  Zustände: $\mathrm{LR}(k)$-*Items*
  Anfangszustand: $[S' \to \bullet S, \varepsilon]$
  Endzustände: $\left\{[B \to \gamma\bullet, x] \mid B \to \gamma \in P, x \in \mathrm{Follow}_k(B)\right\}$
  Übergänge:
  1. $([A \to \alpha \bullet \alpha X\beta, x], X, [A \to \alpha X \bullet \beta, x])$
     für $X \in (N \cup T)$, $A \to \alpha X\beta \in P$
  2. $([A \to \alpha \bullet B\beta, x], \varepsilon, [B \to \bullet\gamma, x'])$
     für $A \to \alpha B\beta$, $B \to \gamma \in P$,
     $x' \in \mathrm{First}_k(\beta) \odot \{x\}$
- Den **kanon. $\mathrm{LR}(k)$-Automaten** $\mathrm{LR}(G, k)$ erhält man aus $c(G, k)$, indem man ε-Übergänge entfernt & Teilmengenkonstruktion anwendet.
  Alternative Konstruktion von $\mathrm{LR}(G, k)$:
  Hilfsfunktion $\delta_\varepsilon^*(q) := q \cup \left\{[B \to \bullet\gamma, x] \mid \exists [A \to \alpha \bullet B'\beta', x'] \in q, \beta \in (N \cup T)^* : B' \to^* B\beta \wedge x \in \mathrm{First}_k(\beta\beta') \odot \{x'\}\right\}$
  Zustände: Mengen von $\mathrm{LR}(k)$-*Items*
  Anfangszustand: $\delta_\varepsilon^*\{[S' \to \bullet S, \varepsilon]\}$
  Endzustände: $\{q \mid \exists A \to \alpha \in P : [A \to \alpha\bullet, x] \in q\}$
  Übergänge: $\delta(q, X) := \delta_\varepsilon^*\left\{[A \to \alpha X \bullet \beta, x] \mid [A \to \alpha \bullet X\beta, x] \in q\right\}$
- 2 mögliche Konflikte:
  - ▷ ***Reduce-Reduce*-Konflikt:**
    $[A \to \gamma\bullet, x], [A' \to \gamma'\bullet, x] \in q$
    mit $A \neq A' \vee \gamma \neq \gamma'$
  - ▷ ***Shift-Reduce*-Konflikt:**
    $[A \to \gamma\bullet, x], [A' \to \alpha \bullet a\beta, y] \in q$
    mit $a \in T$ und $x \in \{a\} \odot \mathrm{First}_k(\beta) \odot \{y\}$
    für einen Zustand $q \in Q$.

  Solche Zustände heißen **$\mathrm{LR}(k)$-ungeeignet**.
- Satz: Eine reduzierte kontextfr. Gr. G ist $\mathrm{LR}(k)$ $\Leftrightarrow$ der kanon. $\mathrm{LR}(k)$-Automat $\mathrm{LR}(G, k)$ keine $\mathrm{LR}(k)$-ungeeigneten Zustände besitzt.
- i. A. hat $\mathrm{LR}(G, k)$ sehr viel mehr Zustände als $\mathrm{LR}(G) = \mathrm{LR}(G, 0)$.
  Man betrachtet darum i. A. Teilklassen von $\mathrm{LR}(k)$-Grammatiken, bei denen man nur $\mathrm{LR}(G)$ benutzt.
- **$\mathrm{LR}(k)$-Parser:**
  - ▷ **goto-Tabelle** kodiert die Zustandsübergänge: $\mathrm{goto}[q, X] := \delta(q, X) \in Q$
  - ▷ **action-Tabelle** beschreibt für jeden Zustand q und mögliche Vorausschau w die erforderliche Aktion. Mögliche Aktionen: *shift*, *reduce* $(A \to \gamma)$, *error*
- **Konfliktlösung:**
  - ▷ ***simple*-$\mathrm{LR}(k)$:**
    - *Reduce-Reduce*-Konflikt:
      ⋮