

# Compilerbau

Markus Lohrey

Universität Siegen

SoSe 2015

Diese Folien sind eine leicht veränderte Fassung der Folien von Axel Simon und Michael Petter (TU München).

**Themengebiet:**

**Einführung**

# Prinzip eines Interpreters:



**Vorteil:** Keine Vorberechnung auf dem Programmtext erforderlich  $\implies$  keine/geringe Startup-Zeit :-)

**Nachteil:** Während der Ausführung werden die Programm-Bestandteile immer wieder analysiert  $\implies$  längere Laufzeit :-)

# Prinzip eines Übersetzters:



## Zwei Phasen:

- Übersetzung des Programm-Texts in ein Maschinen-Programm;
- Ausführung des Maschinen-Programms auf der Eingabe.

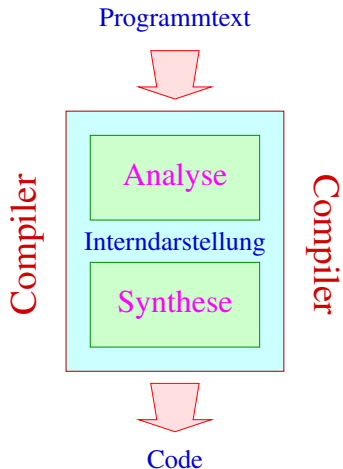
Eine Vorberechnung auf dem Programm gestattet u.a.

- eine geschickte(re) Verwaltung der Variablen;
- Erkennung und Umsetzung globaler Optimierungsmöglichkeiten.

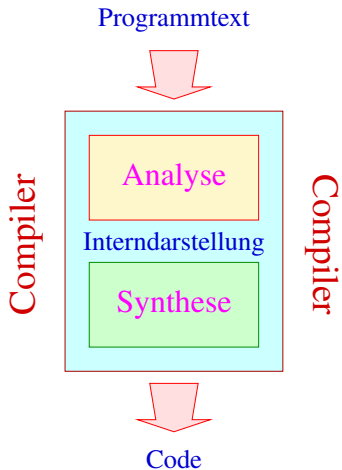
**Nachteil:** Die Übersetzung selbst dauert einige Zeit :-)

**Vorteil:** Die Ausführung des Programme wird effizienter  $\implies$   
lohnt sich bei aufwendigen Programmen und solchen,  
die mehrmals laufen ...

# Übersetzer:

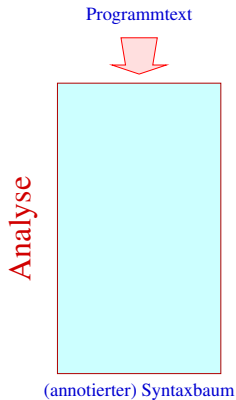


# Übersetzer:



# Übersetzer:

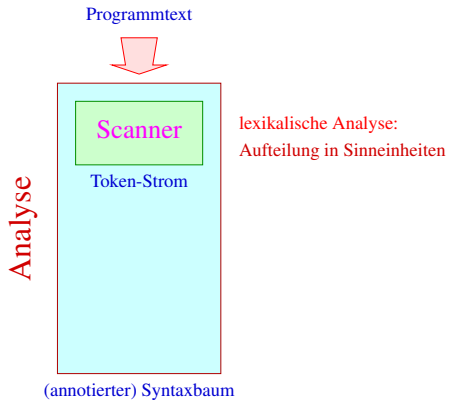
Die Analyse-Phase ist selbst unterteilt in mehrere Schritte





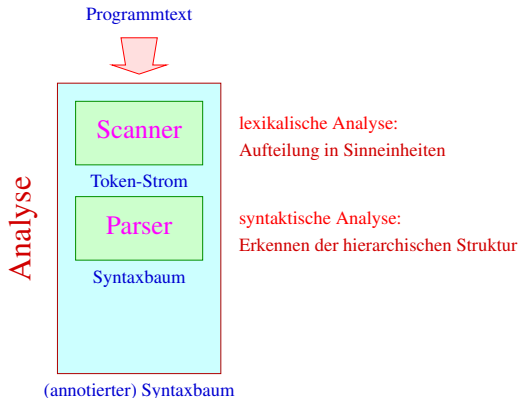
# Übersetzer:

Die Analyse-Phase ist selbst unterteilt in mehrere Schritte



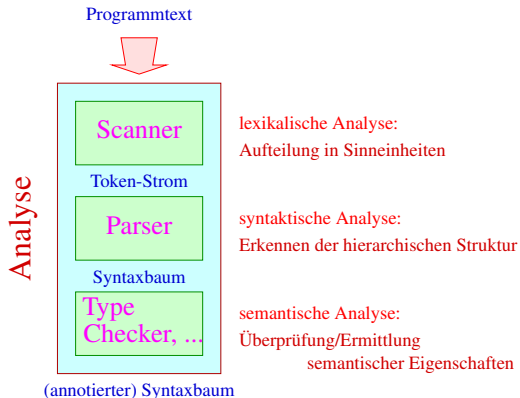
# Übersetzer:

Die Analyse-Phase ist selbst unterteilt in mehrere Schritte



# Übersetzer:

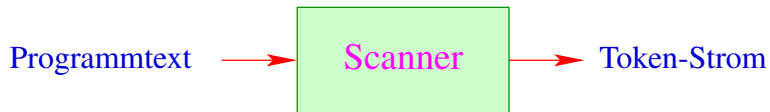
Die Analyse-Phase ist selbst unterteilt in mehrere Schritte



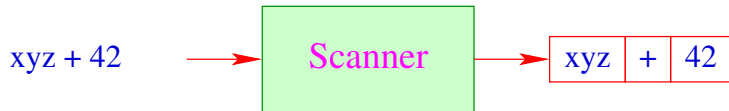
**Themengebiet:**

**Lexikalische Analyse**

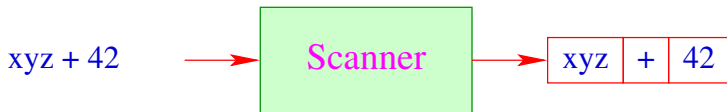
# Die lexikalische Analyse



# Die lexikalische Analyse

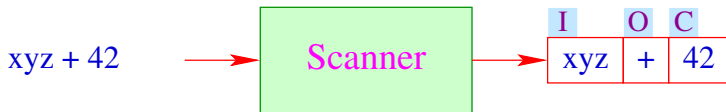


# Die lexikalische Analyse



- Ein **Token** ist eine Folge von Zeichen, die zusammen eine Einheit bilden.
- Tokens werden in **Klassen** zusammen gefasst. Zum Beispiel:
  - **Namen (Identifier)** wie `xyz`, `pi`, ...
  - **Konstanten** wie `42`, `3.14`, `"abc"`, ...
  - **Operatoren** wie `+`, ...
  - **reservierte Worte** wie `if`, `int`, ...

# Die lexikalische Analyse



- Ein **Token** ist eine Folge von Zeichen, die zusammen eine Einheit bilden.
- Tokens werden in **Klassen** zusammen gefasst. Zum Beispiel:
  - **Namen (Identifier)** wie `xyz`, `pi`, ...
  - **Konstanten** wie `42`, `3.14`, `"abc"`, ...
  - **Operatoren** wie `+`, ...
  - **reservierte Worte** wie `if`, `int`, ...



Sind Tokens erst einmal klassifiziert, kann man die Teilwörter **vorverarbeiten**:

- **Wegwerfen** irrelevanter Teile wie **Leerzeichen**, **Kommentaren**,...
- **Aussondern** von **Pragmas**, d.h. Direktiven an den Compiler, die nicht Teil des Programms sind, wie **include**-Anweisungen;
- **Ersetzen** der Token bestimmter Klassen durch ihre Bedeutung / Interndarstellung, etwa bei:
  - **Konstanten**;
  - **Namen**: die typischerweise zentral in einer **Symbol**-Tabelle verwaltet, evt. mit reservierten Worten verglichen (soweit nicht vom Scanner bereits vorgenommen) und gegebenenfalls durch einen Index ersetzt werden.

⇒ **Sieber**

## Diskussion:

- Scanner und Sieber werden i.a. in einer Komponente zusammen gefasst, indem man dem Scanner nach Erkennen eines Tokens gestattet, eine Aktion auszuführen
- Scanner werden i.a. nicht von Hand programmiert, sondern aus einer Spezifikation **generiert**:



## Vorteile:

**Produktivität:** Die Komponente lässt sich **schneller** herstellen

**Korrektheit:** Die Komponente realisiert (beweisbar) die Spezifikation.

**Effizienz:** Der Generator kann die erzeugte Programmkomponente mit den effizientesten Algorithmen ausstatten.

## Vorteile:

**Produktivität:** Die Komponente lässt sich **schneller** herstellen

**Korrektheit:** Die Komponente realisiert (beweisbar) die Spezifikation.

**Effizienz:** Der Generator kann die erzeugte Programmkomponente mit den effizientesten Algorithmen ausstatten.

## Einschränkungen:

- Spezifizieren ist auch **Programmieren** — nur eventuell einfacher
- Generierung statt Implementierung lohnt sich nur für **Routine-Aufgaben**  
... und ist nur für Probleme möglich, die **sehr gut verstanden** sind

... in unserem Fall:



... in unserem Fall:



**Spezifikation von Token-Klassen:** Reguläre Ausdrücke;

**Generierte Implementierung:** Endliche Automaten + X

# Kapitel 1:

## Grundlagen: Reguläre Ausdrücke

## Grundlagen: Reguläre Ausdrücke

- Programmtext benutzt ein endliches **Alphabet**  $\Sigma$  von Eingabe-Zeichen, z.B. ASCII :-)
- Die Menge der Textabschnitte einer Token-Klasse ist i.a. **regulär**.
- Reguläre Sprachen kann man mithilfe **regulärer Ausdrücke** spezifizieren.



## Grundlagen: Reguläre Ausdrücke

- Programmtext benutzt ein endliches **Alphabet**  $\Sigma$  von Eingabe-Zeichen, z.B. ASCII :-)
- Die Menge der Textabschnitte einer Token-Klasse ist i.a. **regulär**.
- Reguläre Sprachen kann man mithilfe **regulärer Ausdrücke** spezifizieren.

Die Menge  $\mathcal{E}_\Sigma$  der (nicht-leeren) **regulären Ausdrücke** ist die kleinste Menge  $\mathcal{E}$  mit:

- $\epsilon \in \mathcal{E}$  ( $\epsilon$  neues Symbol nicht aus  $\Sigma$ );
- $a \in \mathcal{E}$  für alle  $a \in \Sigma$ ;
- $(e_1 \mid e_2), (e_1 \cdot e_2), e_1^* \in \mathcal{E}$  sofern  $e_1, e_2 \in \mathcal{E}$ .



Stephen Kleene, Madison Wisconsin, 1909-1994

## Beispiele:

$$((a \cdot b^*) \cdot a)$$

$$(a \mid b)$$

$$((a \cdot b) \cdot (a \cdot b))$$

## Beispiele:

$$((a \cdot b^*) \cdot a)$$

$$(a \mid b)$$

$$((a \cdot b) \cdot (a \cdot b))$$

### Achtung:

- Wir unterscheiden zwischen Zeichen  $a, 0, |, \dots$  und **Meta-Zeichen**  $(, |, ), \dots$
- Um (hässliche) Klammern zu sparen, benutzen wir **Operator-Präzedenzen**:

$$* > \cdot > |$$

und lassen “.” weg

## Beispiele:

$$\begin{aligned} &((a \cdot b^*) \cdot a) \\ &(a \mid b) \\ &((a \cdot b) \cdot (a \cdot b)) \end{aligned}$$

### Achtung:

- Wir unterscheiden zwischen Zeichen  $a, 0, |, \dots$  und Meta-Zeichen  $(, |, ), \dots$
- Um (hässliche) Klammern zu sparen, benutzen wir Operator-Präzedenzen:

$$* > \cdot > |$$

und lassen “.” weg

- Reale Spezifikations-Sprachen bieten zusätzliche Konstrukte wie:

$$\begin{aligned} e? &\equiv (\epsilon \mid e) \\ e^+ &\equiv (e \cdot e^*) \end{aligned}$$

und verzichten auf “ $\epsilon$ ”

Spezifikationen benötigen eine **Semantik**  
 Im Beispiel:

Spezifikation	Semantik
$ab^*a$	$\{ab^n a \mid n \geq 0\}$
$a \mid b$	$\{a, b\}$
$abab$	$\{abab\}$

Für  $e \in \mathcal{E}_\Sigma$  definieren wir die spezifizierte Sprache  $\llbracket e \rrbracket \subseteq \Sigma^*$   
**induktiv** durch:

$$\begin{aligned}
 \llbracket \epsilon \rrbracket &= \{\epsilon\} \\
 \llbracket a \rrbracket &= \{a\} \\
 \llbracket e^* \rrbracket &= (\llbracket e \rrbracket)^* \\
 \llbracket e_1 \mid e_2 \rrbracket &= \llbracket e_1 \rrbracket \cup \llbracket e_2 \rrbracket \\
 \llbracket e_1 \cdot e_2 \rrbracket &= \llbracket e_1 \rrbracket \cdot \llbracket e_2 \rrbracket
 \end{aligned}$$

## Beachte:

- Die Operatoren  $(\_)^*$ ,  $\cup$ ,  $\cdot$  sind die entsprechenden Operationen auf Wort-Mengen:

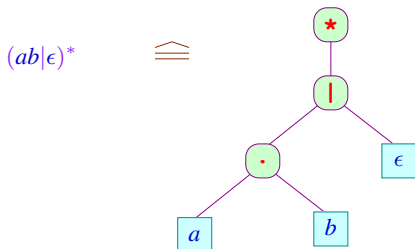
$$\begin{aligned}(L)^* &= \{w_1 \dots w_k \mid k \geq 0, w_i \in L\} \\ L_1 \cdot L_2 &= \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}\end{aligned}$$

## Beachte:

- Die Operatoren  $(\_)*, \cup, \cdot$  sind die entsprechenden Operationen auf Wort-Mengen:

$$\begin{aligned}(L)^* &= \{w_1 \dots w_k \mid k \geq 0, w_i \in L\} \\ L_1 \cdot L_2 &= \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}\end{aligned}$$

- Reguläre Ausdrücke stellen wir intern als **markierte geordnete Bäume** dar:



**Innere Knoten:**  
**Blätter:**

Operator-Anwendungen;  
einzelne Zeichen oder  $\epsilon$ .



## Anwendung:

### Identifizier in Java:

`le = [a-zA-Z_\$]`

`di = [0-9]`

`Id = {le} ({le} | {di})*`

## Anwendung:

### Identifier in Java:

le = [a-zA-Z\_\\$]

di = [0-9]

Id = {le} ({le} | {di})\*

Float = {di}\* (\.{di}|{di}\.) {di}\* ((e|E) (\+|\-)?{di}+)?

## Anwendung:

### Identifizier in Java:

`le = [a-zA-Z_\$]`

`di = [0-9]`

`Id = {le} ({le} | {di})*`

`Float = {di}* (\.{di}|{di}\.) {di}* ((e|E) (\+|\-)?{di}+)?`

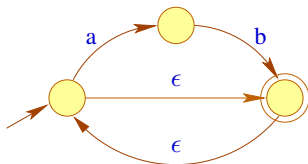
### Bemerkungen:

- “le” und “di” sind **Zeichenklassen**.
- **Definierte Namen** werden in “{”, “}” eingeschlossen.
- Zeichen werden von **Meta-Zeichen** durch “\” unterschieden.

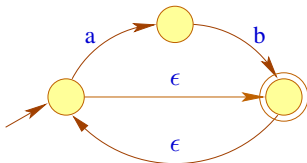
# **Kapitel 2:**

## **Grundlagen: Endliche Automaten**

## Beispiel:



## Beispiel:



**Knoten:** Zustände;

**Kanten:** Übergänge;

**Beschriftungen:** konsumierter Input



Michael O. Rabin, Stanford  
University



Dana S. Scott, Carnegie Mellon  
University, Pittsburgh

## Definition

Formal ist ein nicht-deterministischer endlicher Automat mit  $\epsilon$ -Übergängen ( $\epsilon$ -NFA) ein Tupel  $A = (Q, \Sigma, \delta, I, F)$  wobei:

- $Q$  eine endliche Menge von Zuständen;
- $\Sigma$  ein endliches Eingabe-Alphabet;
- $I \subseteq Q$  die Menge der Anfangszustände;
- $F \subseteq Q$  die Menge der Endzustände und
- $\delta$  die Menge der Übergänge (die Übergangs-Relation) ist.



## Definition

Formal ist ein nicht-deterministischer endlicher Automat mit  $\epsilon$ -Übergängen ( $\epsilon$ -NFA) ein Tupel  $A = (Q, \Sigma, \delta, I, F)$  wobei:

- $Q$  eine endliche Menge von Zuständen;
- $\Sigma$  ein endliches Eingabe-Alphabet;
- $I \subseteq Q$  die Menge der Anfangszustände;
- $F \subseteq Q$  die Menge der Endzustände und
- $\delta$  die Menge der Übergänge (die Übergangs-Relation) ist.

Für  $\epsilon$ -NFAs ist:

$$\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$$

## Definition

Formal ist ein nicht-deterministischer endlicher Automat mit  $\epsilon$ -Übergängen ( $\epsilon$ -NFA) ein Tupel  $A = (Q, \Sigma, \delta, I, F)$  wobei:

- $Q$  eine endliche Menge von Zuständen;
- $\Sigma$  ein endliches Eingabe-Alphabet;
- $I \subseteq Q$  die Menge der Anfangszustände;
- $F \subseteq Q$  die Menge der Endzustände und
- $\delta$  die Menge der Übergänge (die Übergangs-Relation) ist.

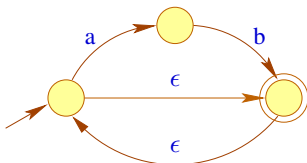
Für  $\epsilon$ -NFAs ist:

$$\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$$

- Gibt es keine  $\epsilon$ -Übergänge  $(p, \epsilon, q)$ , ist  $A$  ein NFA.
- Ist  $\delta : Q \times \Sigma \rightarrow Q$  eine Funktion und  $\#I = 1$ , heißt  $A$  deterministisch (DFA).

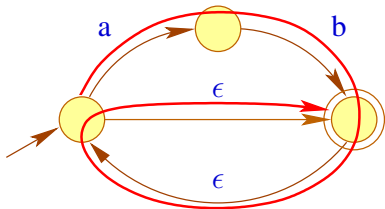
# Akzeptierung

- Berechnungen sind Pfade im Graphen.
- akzeptierende Berechnungen führen von  $I$  nach  $F$ .
- Ein akzeptiertes Wort ist die Beschriftung eines akzeptierenden Pfades ...



# Akzeptierung

- Berechnungen sind Pfade im Graphen.
- akzeptierende Berechnungen führen von  $I$  nach  $F$ .
- Ein akzeptiertes Wort ist die Beschriftung eines akzeptierenden Pfades ...



- Dazu definieren wir den **transitiven Abschluss**  $\delta^*$  von  $\delta$  als kleinste Menge  $\delta'$  mit  $(x \in \Sigma \cup \{\epsilon\}, w \in \Sigma^*)$ :

$$(p, \epsilon, p) \in \delta' \quad \text{und} \\ (p, xw, q) \in \delta' \quad \text{sofern} \quad (p, x, p_1) \in \delta \quad \text{und} \quad (p_1, w, q) \in \delta'.$$

$\delta^*$  beschreibt für je zwei Zustände, mit welchen Wörtern man vom einen zum andern kommt

- Die Menge aller akzeptierten Worte, d.h. die von  $A$  **akzeptierte Sprache** können wir kurz beschreiben als:

$$\mathcal{L}(A) = \{w \in \Sigma^* \mid \exists i \in I, f \in F : (i, w, f) \in \delta^*\}$$

# Umwandlung von Regex in NFA

## Satz:

Für jeden regulären Ausdruck  $e$  kann (in linearer Zeit) ein  $\epsilon$ -NFA konstruiert werden, der die Sprache  $\llbracket e \rrbracket$  akzeptiert.

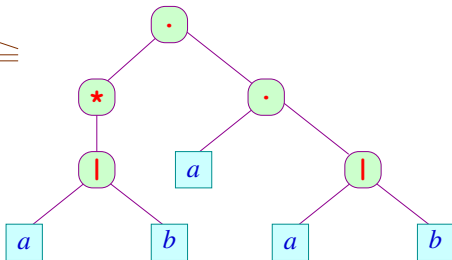
## Idee:

Der Automat verfolgt (konzeptionell mithilfe einer Marke “ $\bullet$ ”), wohin man in  $e$  mit der Eingabe  $w$  gelangen kann.

## Beispiel:

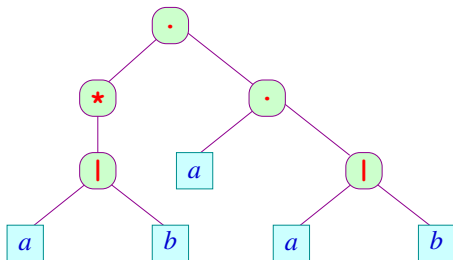
$(a|b)^* a(a|b)$

$\cong$



## Beispiel:

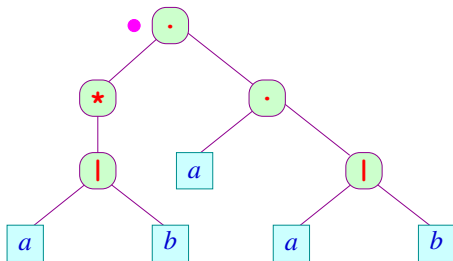
$w = bbaa$  :





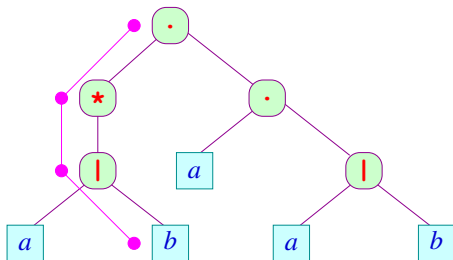
## Beispiel:

$w = bbaa$  :



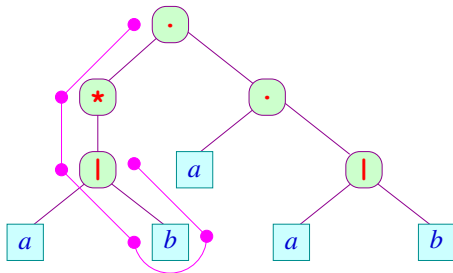
# Beispiel:

$w = bbaa$  :



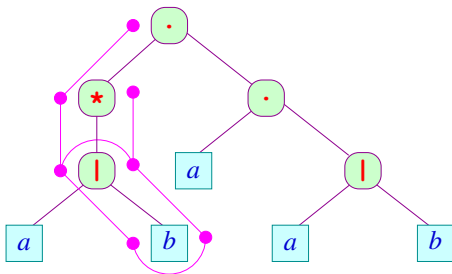
# Beispiel:

$w = bbaa$  :



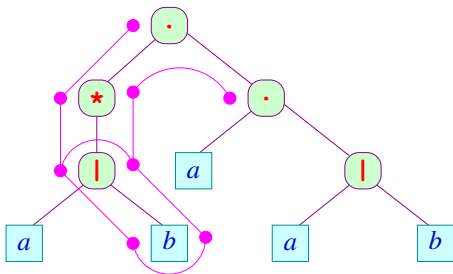
# Beispiel:

$w = bbaa$  :



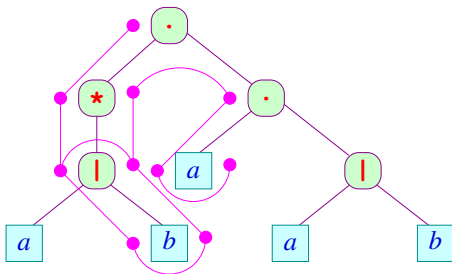
# Beispiel:

$w = bbaa$  :



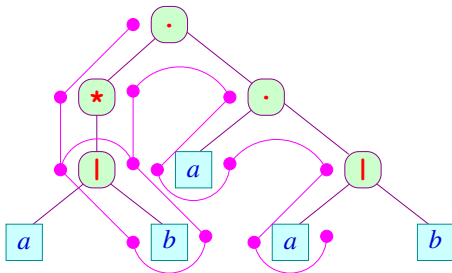
# Beispiel:

$w = bbaa$  :



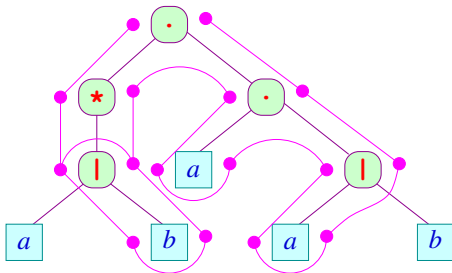
# Beispiel:

$w = bbaa$  :



# Beispiel:

$w = bbaa$  :



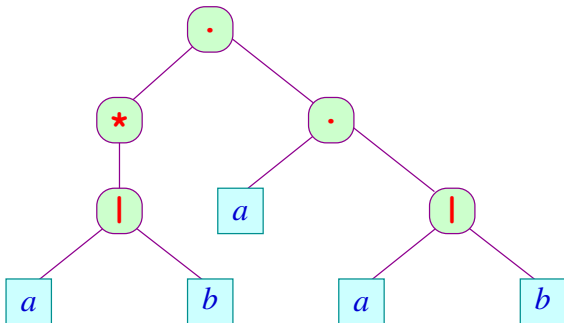


## Beachte:

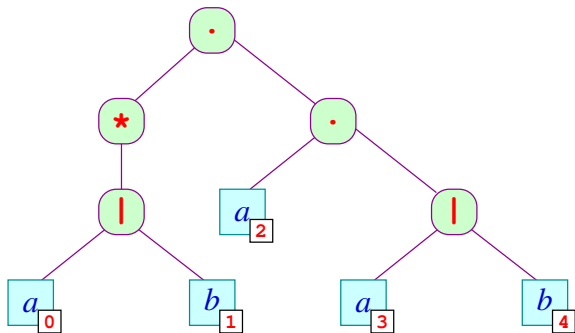
- Gelesen wird nur an den Blättern.
- Die Navigation im Baum erfolgt ohne Lesen, d.h. mit  $\epsilon$ -Übergängen.
- Für eine formale Konstruktion müssen wir die Knoten im Baum **bezeichnen**.
- Dazu benutzen wir (hier) einfach den dargestellten **Teilausdruck**
- Leider gibt es eventuell mehrere gleiche Teilausdrücke

⇒ Wir numerieren die Blätter durch ...

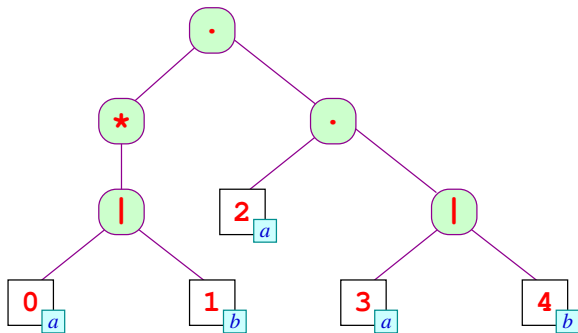
... im Beispiel:



... im Beispiel:



... im Beispiel:



## Die Konstruktion:

**Zustände:**  $\bullet r$ ,  $r \bullet$   $r$  Knoten von  $e$ ;

**Anfangszustand:**  $\bullet e$ ;

**Endzustand:**  $e \bullet$ ;

**Übergangsrelation:** Für Blätter  $r \equiv \boxed{i \mid x}$  benötigen wir:  $(\bullet r, x, r \bullet)$ .

Die übrigen Übergänge sind:

$r$	Übergänge
$r_1 \mid r_2$	$(\bullet r, \epsilon, \bullet r_1)$ $(\bullet r, \epsilon, \bullet r_2)$ $(r_1 \bullet, \epsilon, r \bullet)$ $(r_2 \bullet, \epsilon, r \bullet)$
$r_1 \cdot r_2$	$(\bullet r, \epsilon, \bullet r_1)$ $(r_1 \bullet, \epsilon, \bullet r_2)$ $(r_2 \bullet, \epsilon, r \bullet)$

$r$	Übergänge
$r_1^*$	$(\bullet r, \epsilon, r \bullet)$ $(\bullet r, \epsilon, \bullet r_1)$ $(r_1 \bullet, \epsilon, \bullet r_1)$ $(r_1 \bullet, \epsilon, r \bullet)$
$r_1?$	$(\bullet r, \epsilon, r \bullet)$ $(\bullet r, \epsilon, \bullet r_1)$ $(r_1 \bullet, \epsilon, r \bullet)$

## Diskussion:

- Die meisten Übergänge dienen dazu, im Ausdruck zu navigieren
- Der Automat ist i.a. nichtdeterministisch

## Diskussion:

- Die meisten Übergänge dienen dazu, im Ausdruck zu navigieren
- Der Automat ist i.a. nichtdeterministisch

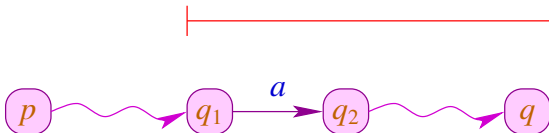


## Strategie:

- 1 Beseitigung der  $\epsilon$ -Übergänge;
- 2 Beseitigung des Nichtdeterminismus

## Beseitigung von $\epsilon$ -Übergängen:

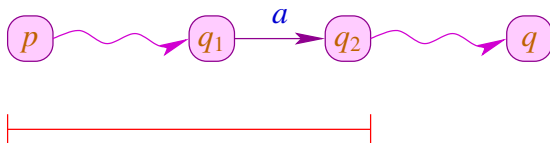
Zwei einfache Ansätze:





## Beseitigung von $\epsilon$ -Übergängen:

Zwei einfache Ansätze:



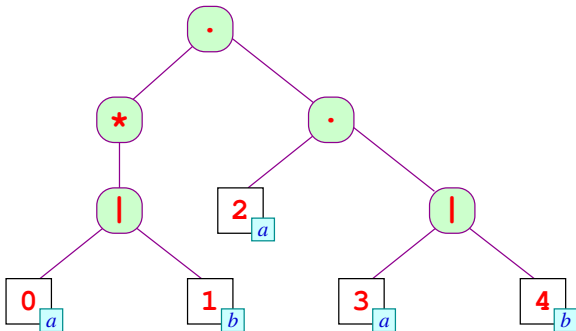
Wir benutzen hier den zweiten Ansatz.

Zur Konstruktion von Parsern werden wir später den ersten benutzen

# 1. Schritt:

$\text{empty}[r] = t$  gdw.  $\epsilon \in [r]$

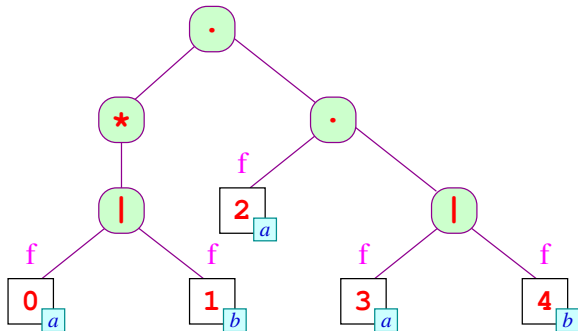
... im Beispiel:



# 1. Schritt:

$\text{empty}[r] = t$  gdw.  $\epsilon \in [r]$

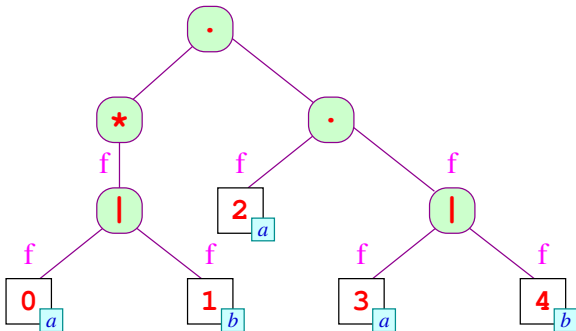
... im Beispiel:



# 1. Schritt:

$\text{empty}[r] = t$  gdw.  $\epsilon \in [r]$

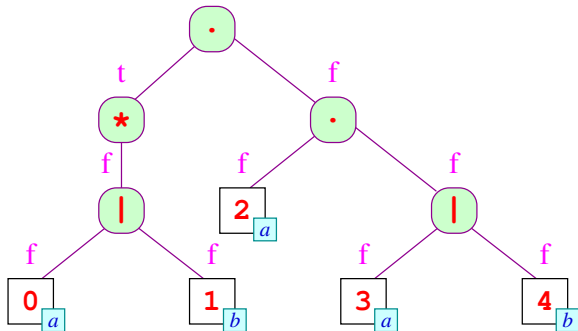
... im Beispiel:



# 1. Schritt:

$\text{empty}[r] = t$  gdw.  $\epsilon \in [r]$

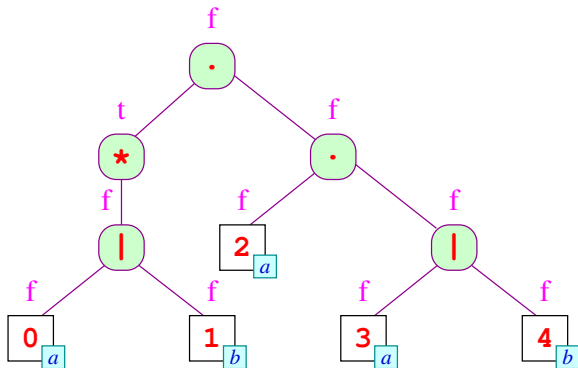
... im Beispiel:



# 1. Schritt:

$\text{empty}[r] = t$  gdw.  $\epsilon \in [r]$

... im Beispiel:



## Implementierung:

### DFS *post-order* Traversierung

Für Blätter  $r \equiv \boxed{i \mid x}$  ist  $\text{empty}[r] = (x \equiv \epsilon)$ .

Andernfalls:

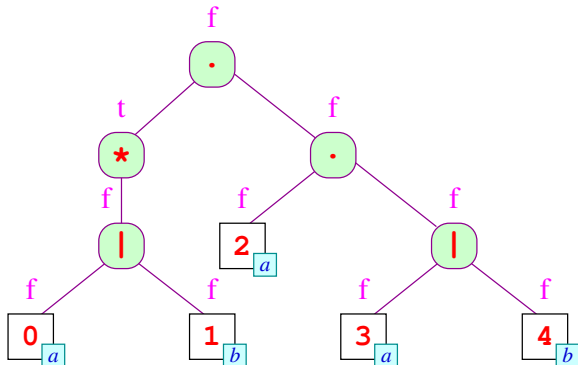
$$\begin{aligned}\text{empty}[r_1 \mid r_2] &= \text{empty}[r_1] \vee \text{empty}[r_2] \\ \text{empty}[r_1 \cdot r_2] &= \text{empty}[r_1] \wedge \text{empty}[r_2] \\ \text{empty}[r_1^*] &= t \\ \text{empty}[r_1?] &= t\end{aligned}$$

## 2. Schritt:

Die Menge erster Blätter:

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

... im Beispiel:



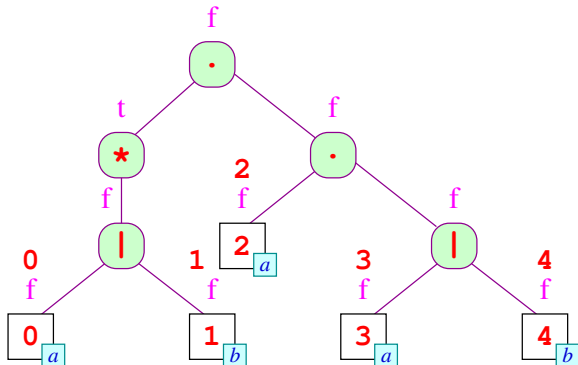


## 2. Schritt:

Die Menge erster Blätter:

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

... im Beispiel:



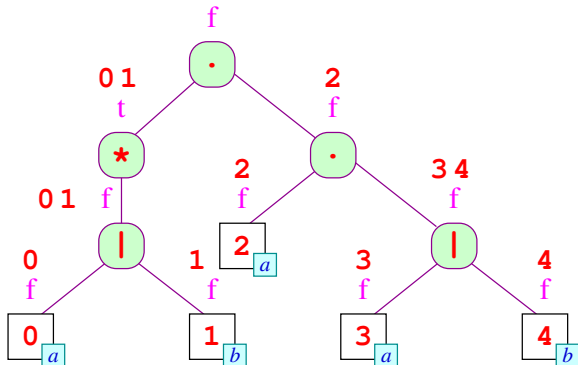


## 2. Schritt:

Die Menge erster Blätter:

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

... im Beispiel:

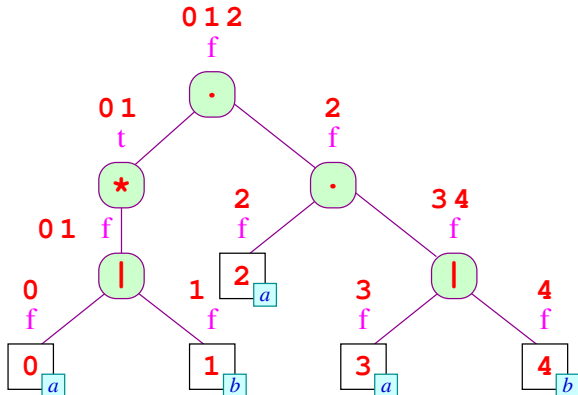


## 2. Schritt:

Die Menge erster Blätter:

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

... im Beispiel:



## Implementierung:

DFS **post-order** Traversierung

Für Blätter  $r \equiv \boxed{i \mid x}$  ist  $\text{first}[r] = \{i \mid x \neq \epsilon\}$ .

Andernfalls:

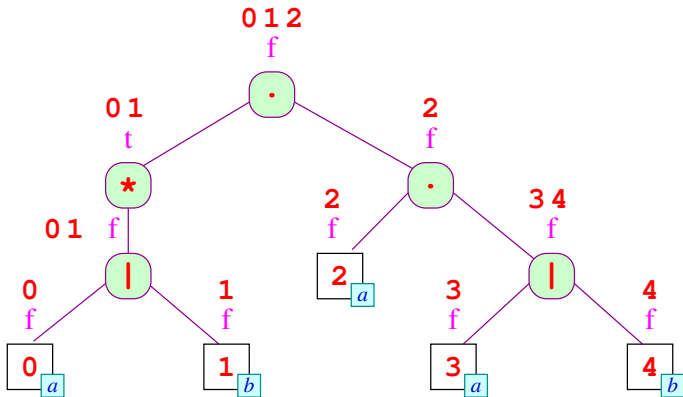
$$\begin{aligned} \text{first}[r_1 \mid r_2] &= \text{first}[r_1] \cup \text{first}[r_2] \\ \text{first}[r_1 \cdot r_2] &= \begin{cases} \text{first}[r_1] \cup \text{first}[r_2] & \text{falls } \text{empty}[r_1] = t \\ \text{first}[r_1] & \text{falls } \text{empty}[r_1] = f \end{cases} \\ \text{first}[r_1^*] &= \text{first}[r_1] \\ \text{first}[r_1?] &= \text{first}[r_1] \end{aligned}$$

### 3. Schritt:

Die Menge nächster Blätter:

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

... im Beispiel:

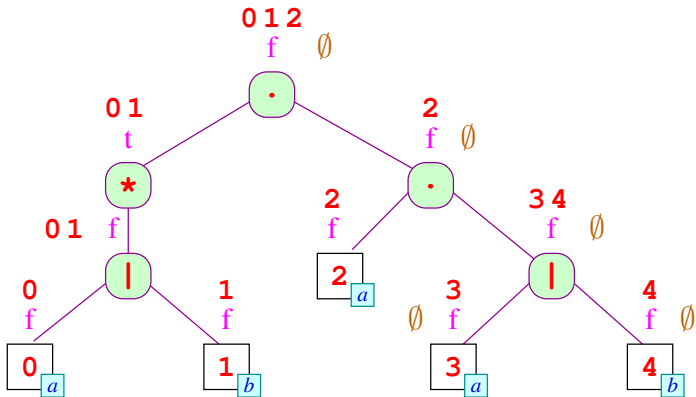


### 3. Schritt:

Die Menge nächster Blätter:

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

... im Beispiel:

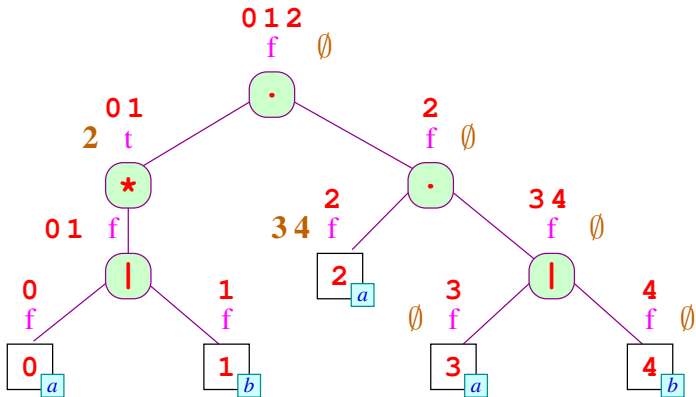


### 3. Schritt:

Die Menge nächster Blätter:

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

... im Beispiel:



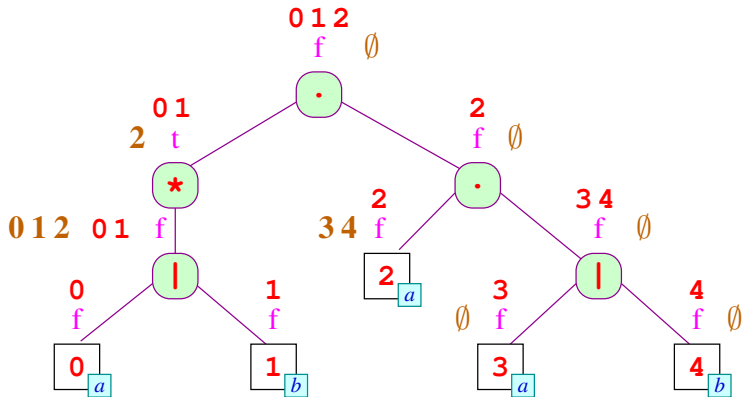


### 3. Schritt:

Die Menge nächster Blätter:

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

... im Beispiel:

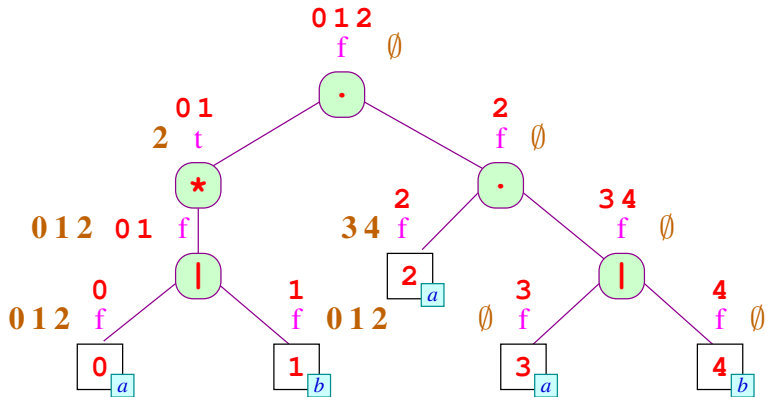


### 3. Schritt:

Die Menge nächster Blätter:

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

... im Beispiel:



## Implementierung:

### DFS pre-order Traversierung

Für die Wurzel haben wir:

$$\text{next}[e] = \emptyset$$

Ansonsten machen wir eine Fallunterscheidung über den **Kontext**:

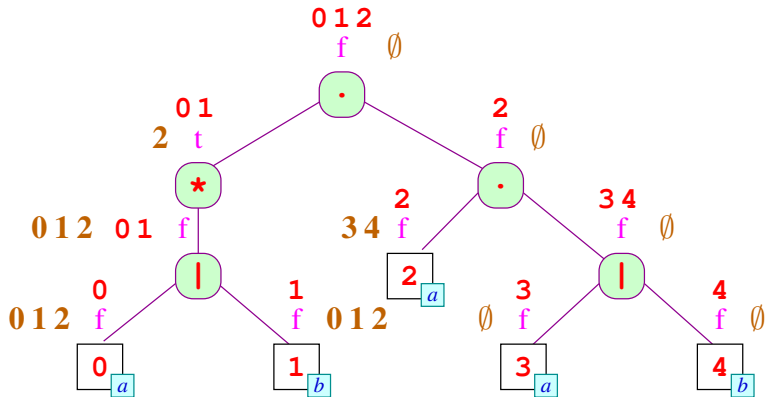
$r$	Regeln
$r_1 \mid r_2$	$\text{next}[r_1] = \text{next}[r]$ $\text{next}[r_2] = \text{next}[r]$
$r_1 \cdot r_2$	$\text{next}[r_1] = \begin{cases} \text{first}[r_2] \cup \text{next}[r] & \text{falls } \text{empty}[r_2] = t \\ \text{first}[r_2] & \text{falls } \text{empty}[r_2] = f \end{cases}$ $\text{next}[r_2] = \text{next}[r]$
$r_1^*$	$\text{next}[r_1] = \text{first}[r_1] \cup \text{next}[r]$
$r_1?$	$\text{next}[r_1] = \text{next}[r]$

## 4. Schritt:

Die Menge letzter Blätter:

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

... im Beispiel:

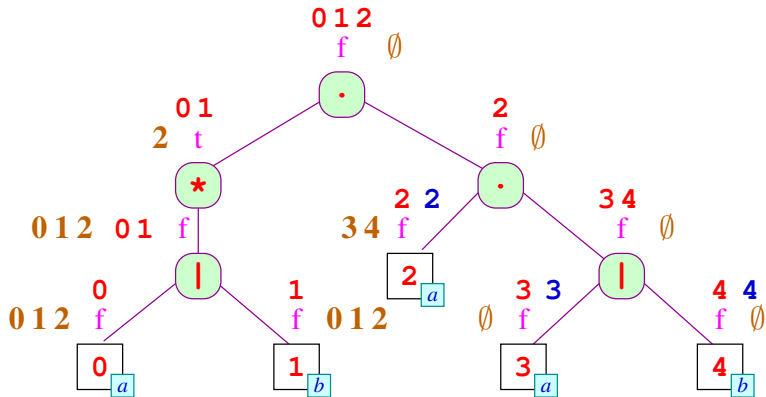


## 4. Schritt:

Die Menge letzter Blätter:

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

... im Beispiel:

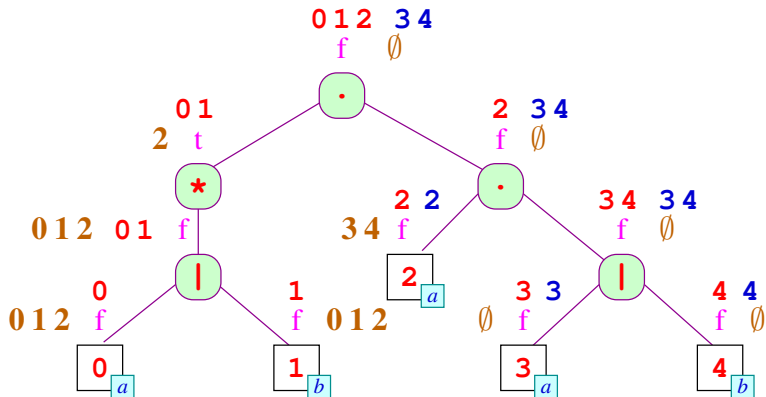


## 4. Schritt:

Die Menge letzter Blätter:

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

... im Beispiel:



## Implementierung:

### DFS post-order Traversierung

Für Blätter  $r \equiv \boxed{i \mid x}$  ist  $\text{last}[r] = \{i \mid x \neq \epsilon\}$ .

Andernfalls:

$$\begin{aligned} \text{last}[r_1 \mid r_2] &= \text{last}[r_1] \cup \text{last}[r_2] \\ \text{last}[r_1 \cdot r_2] &= \begin{cases} \text{last}[r_1] \cup \text{last}[r_2] & \text{falls } \text{empty}[r_2] = t \\ \text{last}[r_2] & \text{falls } \text{empty}[r_2] = f \end{cases} \\ \text{last}[r_1^*] &= \text{last}[r_1] \\ \text{last}[r_1?] &= \text{last}[r_1] \end{aligned}$$

## Integration:

**Zustände:**  $\{q_0\} \uplus \{i \mid i \text{ Blatt in } e, \text{ das nicht mit } \varepsilon \text{ beschriftet ist.}\}$

**Startzustand:**  $q_0$

**Endzustände:** Falls  $\text{empty}[e] = f$ , dann  $\text{last}[e]$ .

Andernfalls:  $\{q_0\} \uplus \text{last}[e]$ .

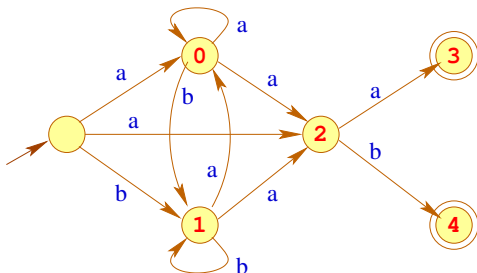
**Übergänge:**  $(q_0, a, i)$  falls  $i \in \text{first}[e]$  und  $i$  mit  $a$  beschriftet ist;

$(i, a, i')$  falls  $i' \in \text{next}[i]$  und  $i'$  mit  $a$  beschriftet ist.

Den resultierenden Automaten bezeichnen wir mit  $A_e$ .

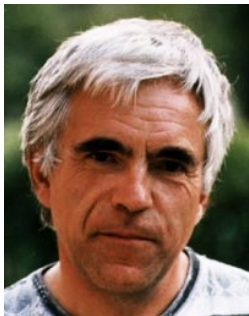


## ... im Beispiel:

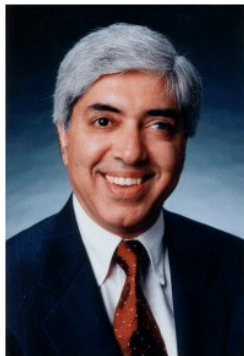


### Bemerkung:

- Die Konstruktion heißt auch **Berry-Sethi-** oder **Glushkow-**Konstruktion.
- Sie wird in **XML** zur Definition von **Content Models** benutzt
- Das Ergebnis ist vielleicht nicht, was wir erwartet haben ...

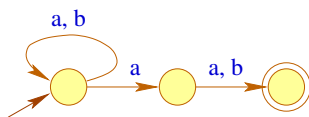


Gerard Berry, Esterel  
Technologies



Ravi Sethi, Research VR, Lucent  
Technologies

## Der erwartete Automat:

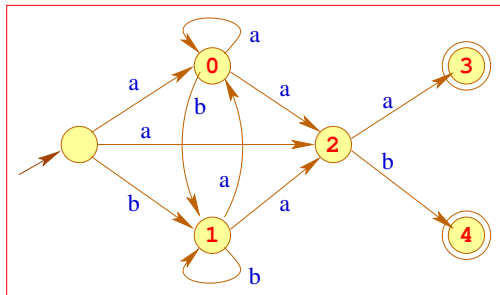


### Bemerkung:

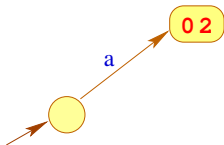
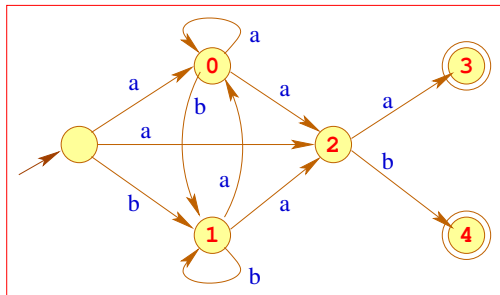
- Im Berry-Sethi-Automat haben alle in einen Zustand eingehenden Kanten die gleiche Beschriftung.
- Aber: Der Berry-Sethi-Automat ist nichtdeterministisch.
- Wir benötigen aber **deterministische** Automaten.

⇒ Teilmengen-Konstruktion

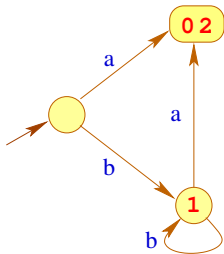
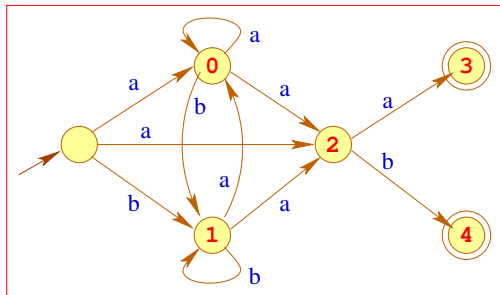
... im Beispiel:



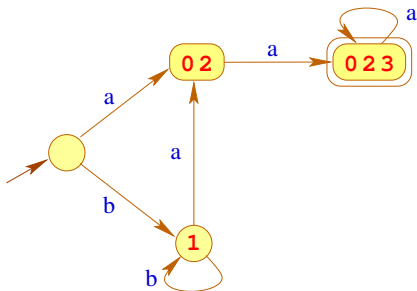
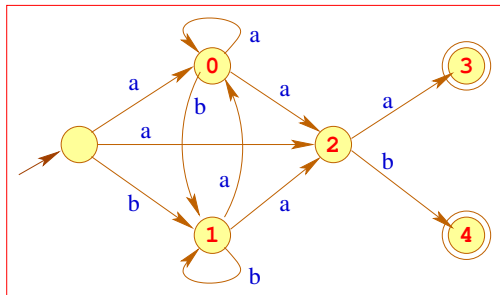
... im Beispiel:



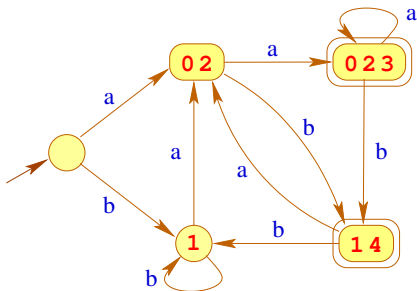
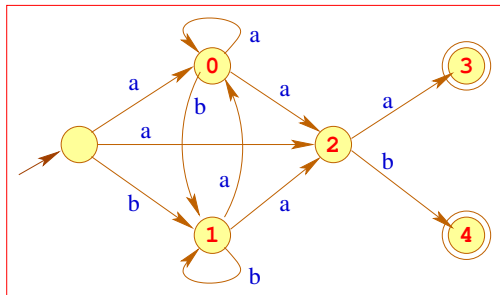
... im Beispiel:



... im Beispiel:



... im Beispiel:





# Teilmengenkonstruktion

## Satz:

Zu jedem nichtdeterministischen Automaten  $A = (Q, \Sigma, \delta, I, F)$  kann ein deterministischer Automat  $\mathcal{P}(A)$  konstruiert werden mit

$$\mathcal{L}(A) = \mathcal{L}(\mathcal{P}(A))$$

# Teilmengenkonstruktion

## Satz:

Zu jedem nichtdeterministischen Automaten  $A = (Q, \Sigma, \delta, I, F)$  kann ein deterministischer Automat  $\mathcal{P}(A)$  konstruiert werden mit

$$\mathcal{L}(A) = \mathcal{L}(\mathcal{P}(A))$$

## Konstruktion:

**Zustände:** Teilmengen von  $Q$ ;

**Anfangszustände:**  $\{I\}$ ;

**Endzustände:**  $\{Q' \subseteq Q \mid Q' \cap F \neq \emptyset\}$ ;

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

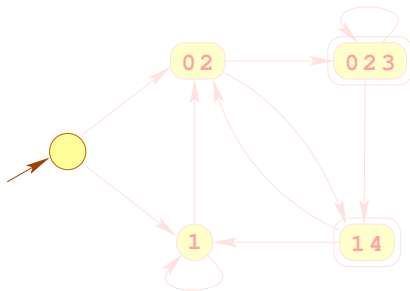
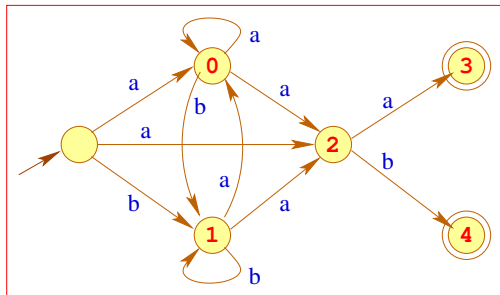
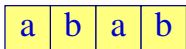
## Achtung:

- Leider gibt es exponentiell viele Teilmengen von  $Q$
- Um nur **nützliche** Teilmengen zu betrachten, starten wir mit der Menge  $Q_P = \{I\}$  und fügen weitere Zustände nur **nach Bedarf** hinzu ...
- d.h., wenn wir sie von einem Zustand in  $Q_P$  aus erreichen können
- Trotz dieser Optimierung kann der Ergebnisautomat **riesig** sein ... was aber in der **Praxis** (so gut wie) nie auftritt

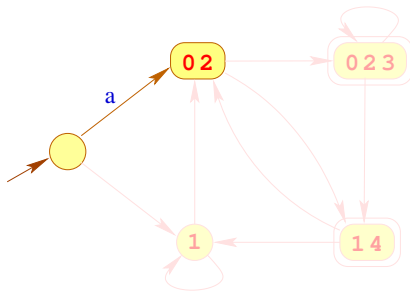
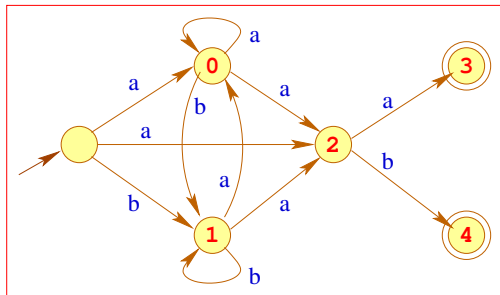
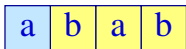
## Achtung:

- Leider gibt es exponentiell viele Teilmengen von  $Q$
- Um nur **nützliche** Teilmengen zu betrachten, starten wir mit der Menge  $Q_P = \{I\}$  und fügen weitere Zustände nur **nach Bedarf** hinzu ...
- d.h., wenn wir sie von einem Zustand in  $Q_P$  aus erreichen können
- Trotz dieser Optimierung kann der Ergebnisautomat **riesig** sein ... was aber in der **Praxis** (so gut wie) nie auftritt
  
- In Tools wie **grep** wird deshalb der **DFA** zu einem regulären Ausdruck nicht aufgebaut !
- Stattdessen werden **während der Abarbeitung der Eingabe** genau die Mengen konstruiert, die für die Eingabe notwendig sind ...

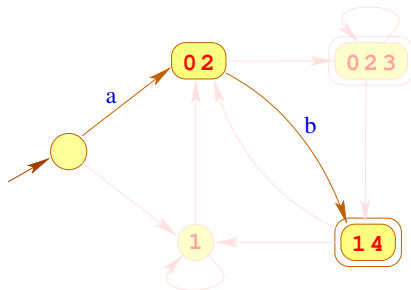
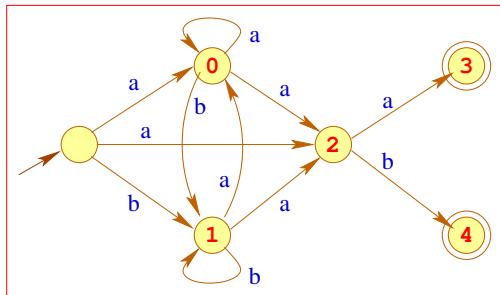
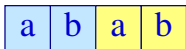
... im Beispiel:



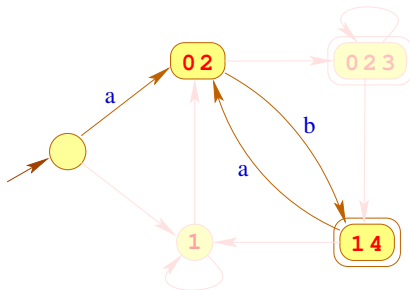
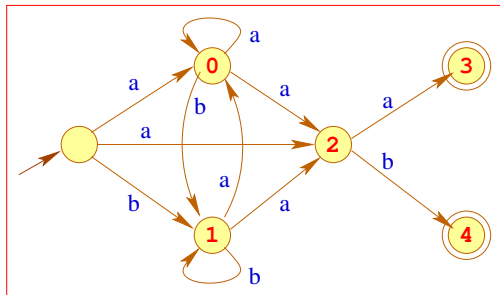
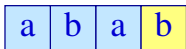
... im Beispiel:



... im Beispiel:



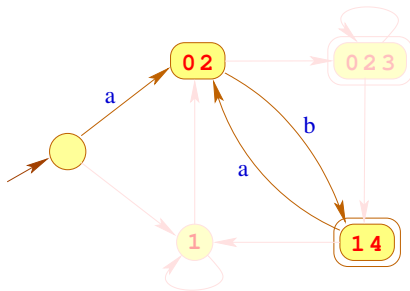
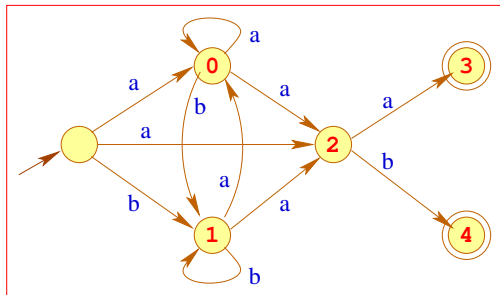
... im Beispiel:





... im Beispiel:

a	b	a	b
---	---	---	---



## Bemerkungen:

- Bei einem Eingabewort der Länge  $n$  werden maximal  $O(n)$  Mengen konstruiert
- Ist eine Menge bzw. eine Kante des DFA einmal konstruiert, heben wir sie in einer Hash-Tabelle auf.
- Bevor wir einen neuen Übergang konstruieren, sehen wir erst nach, ob wir diesen nicht schon haben

## Bemerkungen:

- Bei einem Eingabewort der Länge  $n$  werden maximal  $\mathcal{O}(n)$  Mengen konstruiert
- Ist eine Menge bzw. eine Kante des DFA einmal konstruiert, heben wir sie in einer Hash-Tabelle auf.
- Bevor wir einen neuen Übergang konstruieren, sehen wir erst nach, ob wir diesen nicht schon haben

Zusammenfassend finden wir:

### Satz:

Zu jedem regulären Ausdruck  $e$  kann ein deterministischer Automat  $A = \mathcal{P}(A_e)$  konstruiert werden mit

$$\mathcal{L}(A) = \llbracket e \rrbracket$$

# **Kapitel 3: Design eines Scanners**

# Design eines Scanners

Eingabe (vereinfacht):

eine Menge von Regeln:

$e_1$		{ action <sub>1</sub> }
$e_2$		{ action <sub>2</sub> }
	...	
$e_k$		{ action <sub>k</sub> }

# Design eines Scanners

Eingabe (vereinfacht):            eine Menge von Regeln:

$e_1$	{ action <sub>1</sub> }
$e_2$	{ action <sub>2</sub> }
...	
$e_k$	{ action <sub>k</sub> }

Ausgabe:                            ein Programm, das

- ... von der Eingabe ein **maximales Präfix**  $w$  liest, das  $e_1 \mid \dots \mid e_k$  erfüllt;
- ... das **minimale**  $i$  ermittelt, so dass  $w \in \llbracket e_i \rrbracket$ ;
- ... für  $w$  action <sub>$i$</sub>  ausführt.

Wir gehen im weiteren davon aus, dass  $\epsilon \notin \llbracket e_i \rrbracket$  gilt (macht in der Praxis Sinn: leeres Wort ist normalerweise kein gültiges Token).

# Implementierung:

## Idee:

- Konstruiere den DFA  $\mathcal{P}(A_e) = (Q, \Sigma, \delta, \{q_0\}, F)$  zu dem Ausdruck  $e = (e_1 \mid \dots \mid e_k)$  (Potenzmengenautomat des Berry-Sethi-NFA);
- Beachte:  $\text{last}[e] = \bigcup_{i=1}^k \text{last}[e_i]$ .  
Ausserdem gilt  $q_0 \notin F$ , da  $\epsilon \notin [e_i]$ .
- Definiere die Mengen:

$$F_1 = \{q \in F \mid q \cap \text{last}[e_1] \neq \emptyset\}$$

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

...

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

Dann ist  $\bigcup_{i=1}^k F_i$  die Menge der Endzustände von  $\mathcal{P}(A_e)$ .

- Für Eingabe  $w$  gilt:  $\delta^*(q_0, w) \in F_i$  genau dann wenn der Scanner für  $w$  `actioni` ausführen soll

## Lösung 1.

- Wir verwalten zwei Zeiger  $s$  und  $k$  in das Eingabewort  $w$ .
- In jedem Durchlauf einer **loop**-Schleife wird das nächste Token lokalisiert und die dafür vorgesehene Aktion ausgeführt.
- Dabei zeigt  $s$  stets auf den Anfang des Tokens und  $k$  wird solange nach rechts geschoben, wie der aktuelle Zustand noch  $\neq \emptyset$  ist und das rechte Wortende noch nicht erreicht ist.
- Dabei merken wir uns immer in  $j$  die letzte Position, wo ein Endzustand erreicht wurde.
- Wenn der aktuelle Zustand schließlich  $\emptyset$  ist oder das rechte Wortende erreicht wurde, brechen wir die Suche ab. Das nächste Token ist der Abschnitt  $w[s, j - 1]$  von Position  $s$  bis Position  $j - 1$ .



**Input:** String  $w \in \Sigma^+$

$s := 1; k := 1$

**loop**

$q := q_0; p := \perp$

**while**  $k \leq |w|$  **and**  $\delta(q, w[k]) \neq \emptyset$  **do**

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

**if**  $q \in F$  **then**

$p := q; j := k$

**fi**

**od**

**if**  $p = \perp$  **then return (failure) fi**

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 stop fi**

**pool**

Der Algorithmus auf der vorherigen Folie hat im Worst-Case eine Laufzeit von  $O(|w|^2)$ .

**Beispiel:**

$$\begin{aligned}e_1 &= ab \\e_2 &= (ab)^*c \\w &= (ab)^m\end{aligned}$$

Dann durchläuft der Algorithmus  $m$  mal die **loop**-Schleife.

Im  $i$ -ten Durchlauf läuft der Zeiger  $k$  von Position  $2(i - 1) + 1$  bis zum Wortende.

Also werden (bis auf einen konstanten Faktor)

$$\sum_{i=1}^m (2m - 2(i - 1)) = 2 \sum_{i=1}^m m - i + 1 = 2 \sum_{i=1}^m i = m(m + 1)$$

viele Schritte gemacht.

## Lösung 2. Reps Maximal-Munch-Algorithmus (Reps, 1998)

### Idee:

- Der Algorithmus speichert für jede Position  $i$  in dem Wort und jeden Zustand  $q \in Q$  des Automaten einen Wahrheitswert **fehlversuch** $[q, i]$ .
- Dieses Bit (zu Beginn **false**) wird auf **true** gesetzt, sobald der Algorithmus feststellt, dass von Position  $i$  an beginnend mit Zustand  $q$  kein Endzustand erreicht werden kann.
- Wenn wir bei einer späteren Suche nach einem Token bei Position  $i$  im Zustand  $q$  ankommen, und  $\text{fehlversuch}[q, i] = \mathbf{true}$  gilt, können wir die Suche sofort abbrechen.
- Um das Array **fehlversuch** zu füllen, verwendet der Algorithmus eine Mengenvariable  $S$ , in die potentielle Kandidaten  $\langle q, k \rangle$ , für die **fehlversuch** auf **true** gesetzt werden könnte, eingefügt werden.
- Beachte nochmals:  $q_0$  ist kein Endzustand!

**Input:** String  $w \in \Sigma^+$

$s := 1; k := 1$

**for all**  $q \in Q, i \in [1, |w| + 1]$  **do** fehlversuch[ $q, i$ ] := false **od**  
**loop**

$q := q_0; p := \perp; S := \{\langle q_0, k \rangle\}$

**while**  $k \leq |w|$  **and**  $\delta(q, w[k]) \neq \emptyset$  **do**

**if** fehlversuch[ $q, k$ ] = true **then break fi**

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

**if**  $q \in F$  **then**

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

**else**

$S := S \cup \{\langle q, k \rangle\}$

**fi**

**od**

**if**  $p = \perp$  **then return (failure) fi**

**for all**  $\langle q, k \rangle \in S$  **do** fehlversuch[ $q, k$ ] := true **od**

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 stop fi**

**pool**

Warum benötigt Reps Maximal-Munch-Algorithmus nur lineare Zeit  $O(|w|)$ ?

Der Algorithmus kann nur  $|Q|$  mal eine Position  $k$  in dem Wort besuchen, denn:

- Entweder ein Besuch ist erfolgreich, d.h. wir gelangen später in einen Endzustand,
- oder der Besuch (im Zustand  $q$ ) ist erfolglos, d.h. wir gelangen nicht in einen Endzustand.
- Im ersten Fall gehört  $k$  zu einem lokalisierten Token und wird nicht mehr besucht.
- Im zweiten Fall wird `fehlversuch $[q, k]$`  auf `true` gesetzt, und wir brechen jeden weiteren Besuch von Position  $k$  im Zustand  $q$  sofort ab.

Laufzeit  $O(|Q| \cdot |w|)$ .

$|Q|$  ist in realen Anwendungen wesentlich kleiner als  $|w|$ .

## Erweiterung: Zustände

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

**Beispiel:** Kommentare

Innerhalb eines Kommentars werden Identifier, Konstanten, Kommentare, ... nicht erkannt

Eingabe (verallgemeinert):

eine Menge von Regeln:

```
<state> { e1 { action1 yybegin(state1); }  
          e2 { action2 yybegin(state2); }  
          ...  
          ek { actionk yybegin(statek); }  
        }
```

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

... im Beispiel:

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

## Bemerkungen:

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



# **Kapitel 4: Implementierung von DFAs**

# Implementierung von DFAs

## Aufgaben:

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

Die Klassifizierungsfunktion  $r : Q \rightarrow \mathbb{N}$  wäre bei unserem Automaten  $\mathcal{P}(A_e) = (Q, \Sigma, \delta, \{q_0\}, F)$  (mit  $e = (e_1 \mid \dots \mid e_k)$ ) definiert durch

$$r(q) = \begin{cases} 0 & \text{falls } q \notin F \\ i & \text{falls } q \in F_i \end{cases}$$

# Implementierung von DFAs

## Aufgaben:

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

Die Klassifizierungsfunktion  $r : Q \rightarrow \mathbb{N}$  wäre bei unserem Automaten  $\mathcal{P}(A_e) = (Q, \Sigma, \delta, \{q_0\}, F)$  (mit  $e = (e_1 \mid \dots \mid e_k)$ ) definiert durch

$$r(q) = \begin{cases} 0 & \text{falls } q \notin F \\ i & \text{falls } q \in F_i \end{cases}$$

## Probleme:

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

# Reduktion der Anzahl der Zustände

## Idee: Minimierung

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

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

- Die neuen Zustände sind Äquivalenzklassen der alten Zustände

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

# Reduktion der Anzahl der Zustände

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

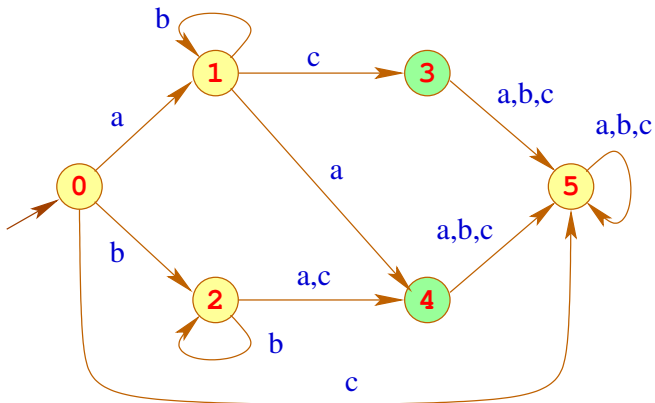
**Idee:**

- Wir nehmen an, **maximal viel** sei äquivalent  
Wir starten mit der Partition:

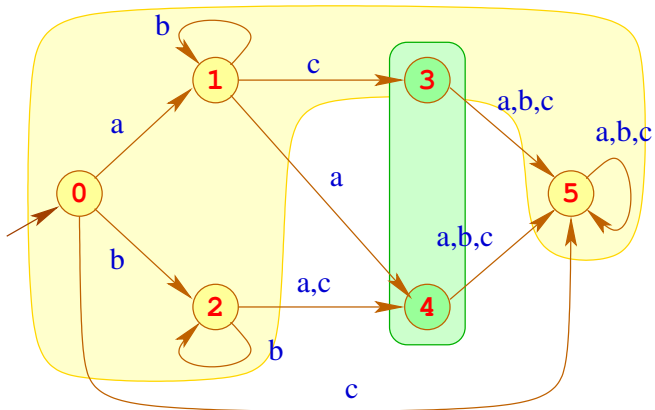
$$\bar{Q} = \{r^{-1}(i) \mid i \in \mathbb{N}, r^{-1}(i) \neq \emptyset\}$$

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

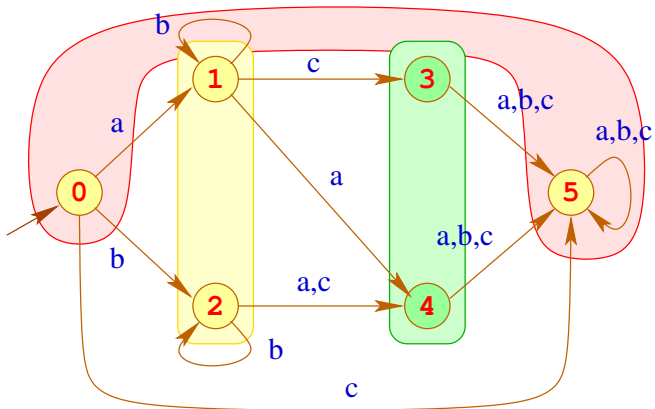
... im Beispiel:



... im Beispiel:

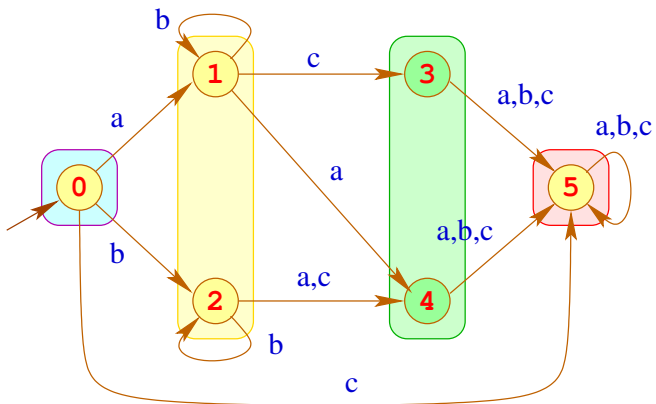


... im Beispiel:





... im Beispiel:



## Bemerkungen:

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

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

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

- Der Ergebnis-Automat ist der **eindeutig bestimmte minimale Automat** für  $\mathcal{L}(A)$
- Eine naive Implementierung erfordert Laufzeit  $\mathcal{O}(n^2)$ .  
Eine raffinierte Verwaltung der Partition liefert ein Verfahren mit Laufzeit  $\mathcal{O}(n \cdot \log(n))$ .



Anil Nerode , Cornell University, Ittaca



John E. Hopcroft, Cornell University, Ittaca

# Reduktion der Tabellengröße

## Problem:

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

# Reduktion der Tabellengröße

## 1. Idee:

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

## Beispiel:

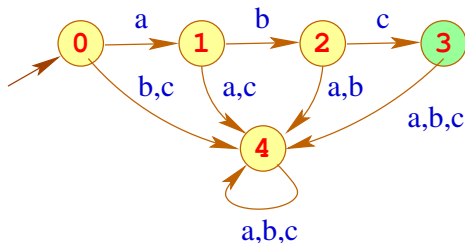
- Der Automat soll deterministisch sein.
- Sind die Klassen der Spezifikation nicht disjunkt, teilt man sie darum in Unterklassen auf, hier in die Klassen `[a-zA-Z_ $]` und `[0-9]`

# Reduktion der Tabellengröße

## 2. Idee:

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

## Beispiel:



## ... die zugehörige Tabelle:

	0	1	2	3	4
<i>a</i>	1	4	4	4	4
<i>b</i>	4	2	4	4	4
<i>c</i>	4	4	3	4	4

### Beobachtung:

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



## ... die zugehörige Tabelle:

	0	1	2	3	4
<i>a</i>	1				
<i>b</i>		2			
<i>c</i>			3		

### Beobachtung:

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

## ... im Beispiel:

	0	1	2
A	1	2	3
valid	a	b	c

- Feld **valid** teilt mit, für welches Element aus  $\Sigma$  der Eintrag gilt
- **Achtung:** I.a. werden die Zeilen nicht so perfekt übereinander passen!  
Dann verschieben wir sie so lange, bis die jeweils nächste in die bisherigen Löcher hineinpasst.
- Darum müssen wir ein zusätzliches Feld **displacement** verwalten, in dem wir uns die Verschiebung merken

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

```
 $\delta(j, a) =$  let  $d = \text{displacement}[a]$   
in if ( $\text{valid}[d + j] \equiv a$ )  
    then  $A[d + j]$   
    else Default
```

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

```
 $\delta(j, a) =$  let  $d = \text{displacement}[a]$   
in if ( $\text{valid}[d + j] \equiv a$ )  
    then  $A[d + j]$   
    else Default
```

## Diskussion:

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

**Themengebiet:**

**Syntaktische Analyse**

# Die syntaktische Analyse



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

# Die syntaktische Analyse



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

## Diskussion:

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





## Diskussion:

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



**Spezifikation der hierarchischen Struktur:** kontextfreie  
Grammatiken

**Generierte Implementierung:** Kellerautomaten + X

# **Kapitel 1:**

## **Grundlagen: Kontextfreie Grammatiken**

## Grundlagen: Kontextfreie Grammatiken

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

## Grundlagen: Kontextfreie Grammatiken

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

### Definition:

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

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



Noam Chomsky, MIT (Guru)



John Backus, IBM (Erfinder von  
Fortran)

# Konventionen

Die Regeln kontextfreier Grammatiken sind von der Form:

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

# Konventionen

Die Regeln kontextfreier Grammatiken sind von der Form:

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

... im Beispiel:

$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

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

# Konventionen

Die Regeln kontextfreier Grammatiken sind von der Form:

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

... im Beispiel:

$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

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

## Konventionen:

In Beispielen ist die Spezifikation der Nichtterminale und Terminale i.a. implizit:

- Nichtterminale sind:  $A, B, C, \dots, \langle \text{exp} \rangle, \langle \text{stmt} \rangle, \dots;$
- Terminale sind:  $a, b, c, \dots, \text{int}, \text{name}, \dots;$



## ... weitere Beispiele:

$S$  →  $\langle \text{stmt} \rangle$   
 $\langle \text{stmt} \rangle$  →  $\langle \text{if} \rangle$  |  $\langle \text{while} \rangle$  |  $\langle \text{rexp} \rangle$ ;  
 $\langle \text{if} \rangle$  → **if** (  $\langle \text{rexp} \rangle$  )  $\langle \text{stmt} \rangle$  **else**  $\langle \text{stmt} \rangle$   
 $\langle \text{while} \rangle$  → **while** (  $\langle \text{rexp} \rangle$  )  $\langle \text{stmt} \rangle$   
 $\langle \text{rexp} \rangle$  → **int** |  $\langle \text{lexp} \rangle$  |  $\langle \text{lexp} \rangle = \langle \text{rexp} \rangle$  | ...  
 $\langle \text{lexp} \rangle$  → **name** | ...

## ... weitere Beispiele:

```
S      → ⟨stmt⟩
⟨stmt⟩ → ⟨if⟩ | ⟨while⟩ | ⟨rexp⟩;
⟨if⟩   → if ( ⟨rexp⟩ ) ⟨stmt⟩ else ⟨stmt⟩
⟨while⟩ → while ( ⟨rexp⟩ ) ⟨stmt⟩
⟨rexp⟩ → int | ⟨lexp⟩ | ⟨lexp⟩ = ⟨rexp⟩ | ...
⟨lexp⟩ → name | ...
```

### Weitere Konventionen:

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

## Weitere Grammatiken:

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

Die beiden Grammatiken beschreiben die gleiche Sprache

## Weitere Grammatiken:

$E \rightarrow E+E^0$		$E * E^1$		$(E)^2$		name <sup>3</sup>		int <sup>4</sup>
$E \rightarrow E+T^0$		$T^1$						
$T \rightarrow T * F^0$		$F^1$						
$F \rightarrow (E)^0$		name <sup>1</sup>		int <sup>2</sup>				

Die beiden Grammatiken beschreiben die gleiche Sprache

# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.  
Die Regeln geben die möglichen Ersetzungsschritte an.  
Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.  
Die Regeln geben die möglichen Ersetzungsschritte an.  
Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

*E*

# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

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

# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$$\begin{array}{l} \underline{E} \rightarrow \underline{E} + T \\ \rightarrow \underline{T} + T \end{array}$$



# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

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

# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

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

# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

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

# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

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

# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

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

# Wortersetzungssysteme

Grammatiken sind **Wortersetzungssysteme**.

Die Regeln geben die möglichen Ersetzungsschritte an.

Eine Folge solcher Ersetzungsschritte heißt auch **Ableitung**.

... im letzten Beispiel:

$\underline{E}$  →  $\underline{E} + T$   
→  $\underline{T} + T$   
→  $T * \underline{F} + T$   
→  $\underline{T} * \text{int} + T$   
→  $\underline{F} * \text{int} + T$   
→  $\text{name} * \text{int} + \underline{T}$   
→  $\text{name} * \text{int} + \underline{F}$   
→  $\text{name} * \text{int} + \text{int}$

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

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

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

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

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

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

### Bemerkungen:

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

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



# Ableitungsbaum

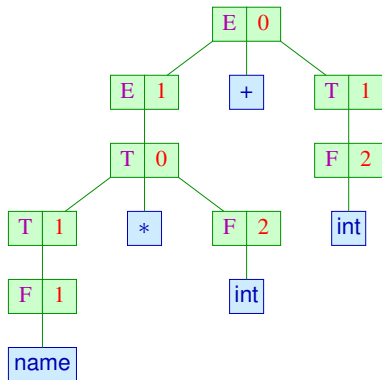
## Achtung:

Die Reihenfolge, in der disjunkte Teile abgeleitet werden, ist unerheblich

Ableitungen eines Symbols stellt man als **Ableitungsbaum** dar:

... im Beispiel:

$\underline{E} \rightarrow^0 \underline{E} + T$   
 $\rightarrow^1 \underline{T} + T$   
 $\rightarrow^0 \underline{T} * \underline{F} + T$   
 $\rightarrow^2 \underline{T} * \text{int} + T$   
 $\rightarrow^1 \underline{F} * \text{int} + T$   
 $\rightarrow^1 \text{name} * \text{int} + \underline{T}$   
 $\rightarrow^1 \text{name} * \text{int} + \underline{F}$   
 $\rightarrow^2 \text{name} * \text{int} + \text{int}$



## Ein Ableitungsbaum für $A \in N$ :

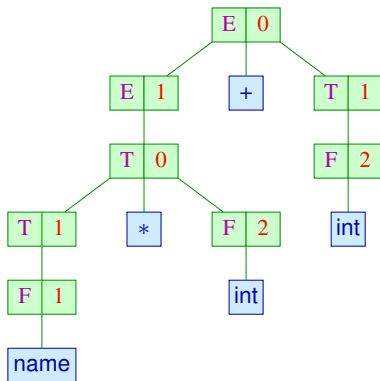
<b>innere Knoten:</b>	Regel-Anwendungen;
<b>Wurzel:</b>	Regel-Anwendung für $A$ ;
<b>Blätter:</b>	Terminale oder $\epsilon$ ;

Die Nachfolger von  $(B, i)$  entsprechen der rechten Seite der Regel

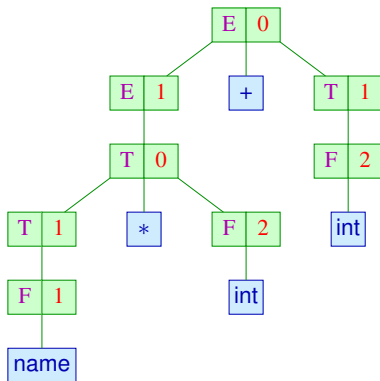
### Beachte:

- Neben beliebiger Ableitungen betrachtet man solche, bei denen stets das **linkste** (bzw. **rechtste**) Vorkommen eines Nichtterminals ersetzt wird.
- Diese heißen **Links-** (bzw. **Rechts-**) Ableitungen und werden durch Index  $L$  bzw.  $R$  gekennzeichnet.
- Links-(bzw. Rechts-) Ableitungen entsprechen einem links-rechts (bzw. rechts-links) **preorder**-DFS-Durchlauf durch den Ableitungsbaum
- **Reverse** Rechts-Ableitungen entsprechen einem links-rechts **postorder**-DFS-Durchlauf durch den Ableitungsbaum

... im Beispiel:



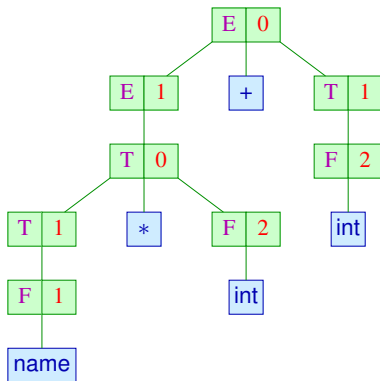
... im Beispiel:



Links-Ableitung:

$(E, 0) (E, 1) (T, 0) (T, 1) (F, 1) (F, 2) (T, 1) (F, 2)$

... im Beispiel:



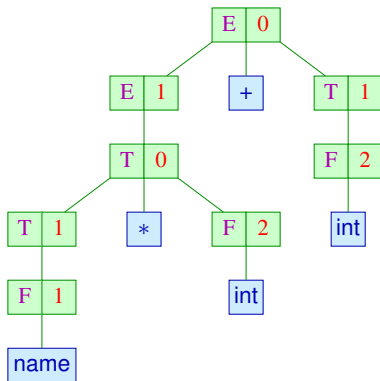
Links-Ableitung:

$(E, 0) (E, 1) (T, 0) (T, 1) (F, 1) (F, 2) (T, 1) (F, 2)$

Rechts-Ableitung:

$(E, 0) (T, 1) (F, 2) (E, 1) (T, 0) (F, 2) (T, 1) (F, 1)$

... im Beispiel:



Links-Ableitung:  $(E, 0) (E, 1) (T, 0) (T, 1) (F, 1) (F, 2) (T, 1) (F, 2)$

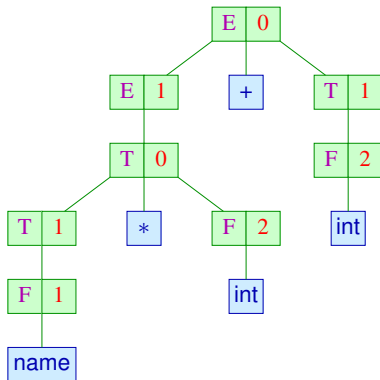
Rechts-Ableitung:  $(E, 0) (T, 1) (F, 2) (E, 1) (T, 0) (F, 2) (T, 1) (F, 1)$

Reverse Rechts-Ableitung:

$(F, 1) (T, 1) (F, 2) (T, 0) (E, 1) (F, 2) (T, 1) (E, 0)$

Die Konkatenation der Blätter des Ableitungsbaums  $t$  bezeichnen wir auch mit  $\text{yield}(t)$ .

... im Beispiel:



liefert die Konkatenation:

`name * int + int`.

# Eindeutige Grammatiken

## Definition:

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

... unsere beiden Grammatiken:

$E \rightarrow E+E^0$	$E * E^1$	$(E)^2$	$\text{name}^3$	$\text{int}^4$
$E \rightarrow E+T^0$	$T^1$			
$T \rightarrow T * F^0$	$F^1$			
$F \rightarrow (E)^0$	$\text{name}^1$	$\text{int}^2$		

Die zweite ist eindeutig, die erste nicht



## Fazit:

- Ein Ableitungsbaum repräsentiert eine mögliche hierarchische Struktur eines Worts.
- Bei Programmiersprachen sind wir nur an Grammatiken interessiert, bei denen die Struktur stets eindeutig ist :-)
- Ableitungsbäume stehen in eins-zu-eins-Korrespondenz mit Links-Ableitungen wie auch (reversen) Rechts-Ableitungen.
- Links-Ableitungen entsprechen einem Topdown-Aufbau des Ableitungsbaums.
- Reverse Rechts-Ableitungen entsprechen einem Bottom-up-Aufbau des Ableitungsbaums.

## **Kapitel 2:**

# **Überflüssige Nichtterminale und Regeln**

# Produktive und erreichbare Nichtterminale

## Definition:

$A \in N$  heißt **produktiv**, falls  $A \rightarrow^* w$  für ein  $w \in T^*$

$A \in N$  heißt **erreichbar**, falls  $S \rightarrow^* \alpha A \beta$  für geeignete  $\alpha, \beta \in (T \cup N)^*$

## Beispiel:

$$\begin{array}{l} S \rightarrow aBB \mid bD \\ A \rightarrow Bc \\ B \rightarrow Sd \mid C \\ C \rightarrow a \\ D \rightarrow BD \end{array}$$

# Produktive und erreichbare Nichtterminale

## Definition:

$A \in N$  heißt **produktiv**, falls  $A \rightarrow^* w$  für ein  $w \in T^*$

$A \in N$  heißt **erreichbar**, falls  $S \rightarrow^* \alpha A \beta$  für geeignete  $\alpha, \beta \in (T \cup N)^*$

## Beispiel:

$$\begin{array}{l} S \rightarrow aBB \mid bD \\ A \rightarrow Bc \\ B \rightarrow Sd \mid C \\ C \rightarrow a \\ D \rightarrow BD \end{array}$$

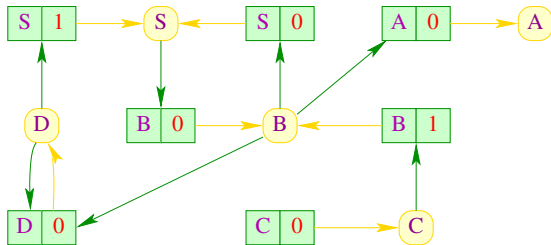
**Produktive Nichtterminale:**  $S, A, B, C$

**Erreichbare Nichtterminale:**  $S, B, C, D$

# And-Or-Graph

Idee für Produktivität: And-Or-Graph für die Grammatik

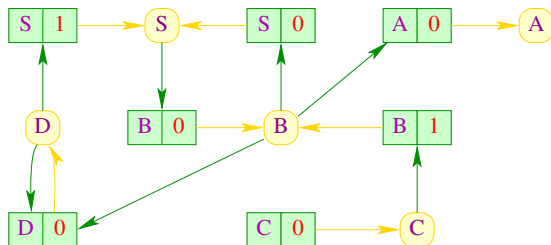
... hier:



# And-Or-Graph

Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



**And-Knoten:** Regeln

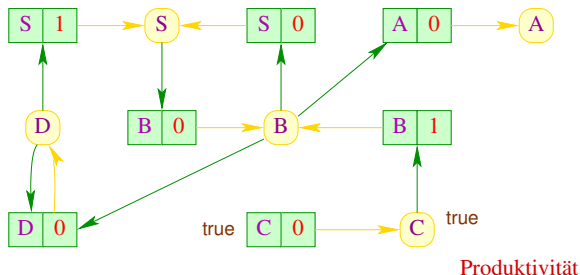
**Or-Knoten:** Nichtterminale

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

# And-Or-Graph

Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



**And-Knoten:** Regeln

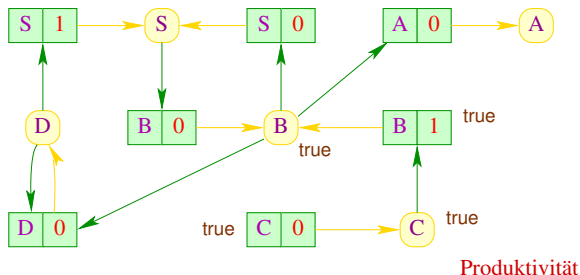
**Or-Knoten:** Nichtterminale

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

# And-Or-Graph

Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



**And-Knoten:** Regeln

**Or-Knoten:** Nichtterminale

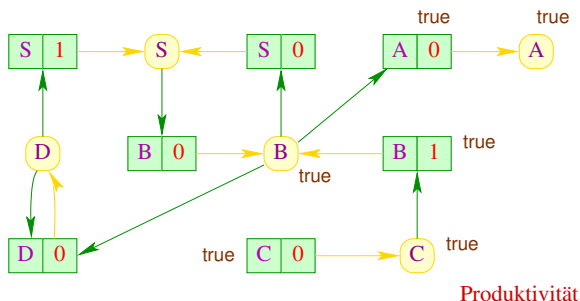
**Kanten:**  $((B, i), B)$  für alle Regeln  $(B, i)$   
 $(A, (B, i))$  falls  $(B, i) \equiv B \rightarrow \alpha_1 A \alpha_2$



# And-Or-Graph

Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



**And-Knoten:** Regeln

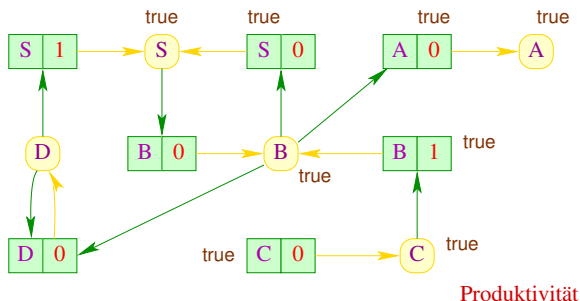
**Or-Knoten:** Nichtterminale

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

# And-Or-Graph

Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



**And-Knoten:** Regeln

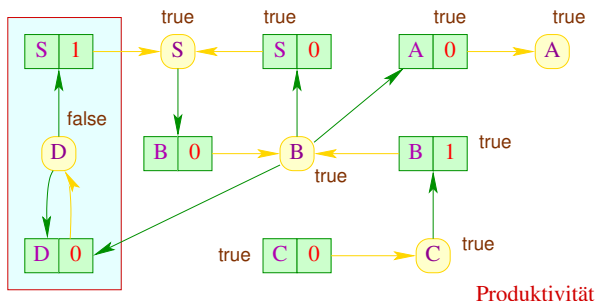
**Or-Knoten:** Nichtterminale

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

# And-Or-Graph

Idee für Produktivität: And-Or-Graph für die Grammatik

... hier:



**And-Knoten:** 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:

```
2N result = ∅;           // Ergebnis-Menge
int count[P];           // Zähler für jede Regel
2P rhs[N];             // Vorkommen in rechten Seiten

forall (A ∈ N) rhs[A] = ∅; // Initialisierung
forall ((A, i) ∈ P) {    //
    count[(A, i)] = 0;   //
    init(A, i);         // Initialisierung von rhs
}                       //
...                     //
```

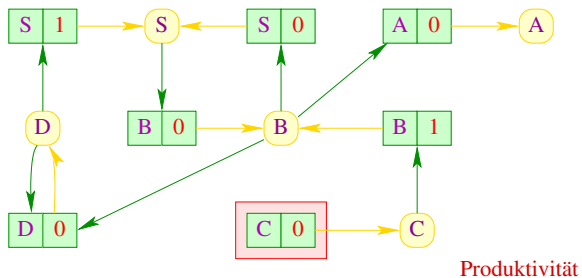
Die Hilfsfunktion **init** geht beim Aufruf **init(A, i)** die rechte Seite der Produktion (A, i) durch. Für jedes Nichtterminal X, dass dabei gesehen wird, wird count[(A, i)] erhöht, und (A, i) in rhs[X] eingefügt.

## Algorithmus (Fortsetzung):

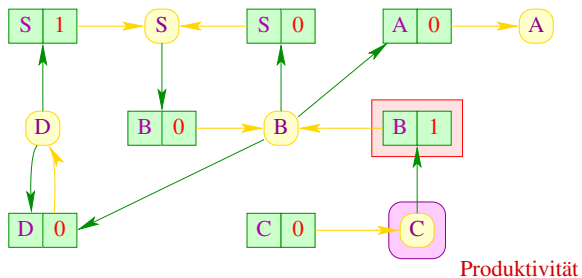
```
... //  
2P W = {r | count[r] = 0}; // Workset  
while (W ≠ ∅) { //  
    (A, i) = extract(W); //  
    if (A ∉ result) { //  
        result = result ∪ {A}; //  
        forall (r ∈ rhs[A]) { //  
            count[r]--; //  
            if (count[r] == 0) W = W ∪ {r}; //  
        } // end of forall  
    } // end of if  
} // end of while
```

Alle Regeln, die in **W** vorkommen, haben auf der linken Seite ein produktives Nichtterminal stehen.

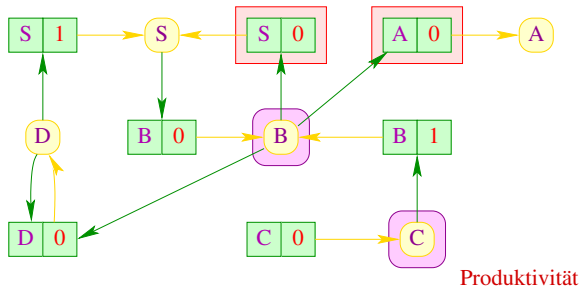
... im Beispiel:



... im Beispiel:

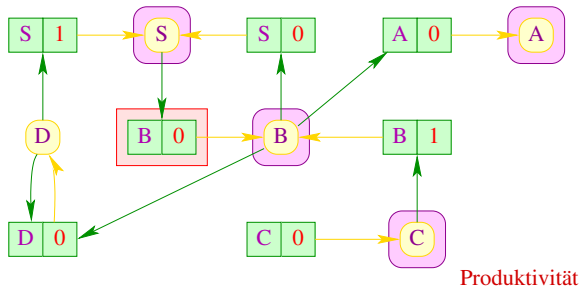


... im Beispiel:

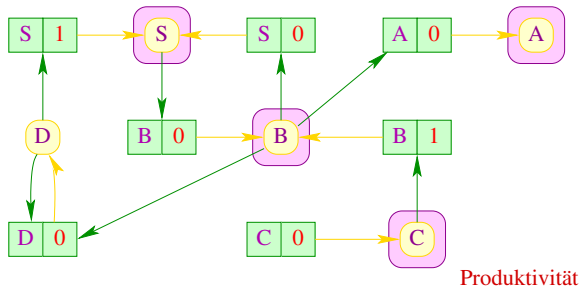




... im Beispiel:



... im Beispiel:



## Laufzeit:

- Die Initialisierung der Datenstrukturen erfordert lineare Laufzeit.
- Jede Regel wird maximal einmal in  $W$  eingefügt.
- Jedes  $A$  wird maximal einmal in  $result$  eingefügt.  
     $\implies$  Laufzeit ist **linear** in der Größe der Grammatik

## Korrektheit:

- Falls  $A$  in der  $j$ -ten Iteration der **while**-Schleife in  $result$  eingefügt, gibt es einen Ableitungsbaum für  $A$  der Höhe maximal  $j - 1$
- Für jeden Ableitungsbaum wird die Wurzel einmal in  $W$  eingefügt

## Diskussion:

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

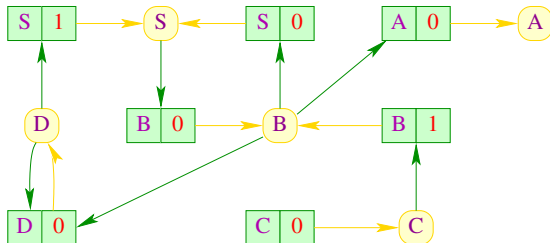
## Diskussion:

- Um den Test  $(A \in \text{result})$  einfach zu machen, repräsentiert man die Menge  $\text{result}$  durch ein **Array**.
- **W** wie auch die Mengen  $\text{rhs}[A]$  wird man dagegen als **Listen** repräsentieren
- Der Algorithmus funktioniert auch, um **kleinste** Lösungen von **Booleschen** Ungleichungssystemen zu bestimmen
- Die Ermittlung der produktiven Nichtterminale kann benutzt werden, um festzustellen, ob  $\mathcal{L}(G) \neq \emptyset$  ist ( $\rightarrow$  **Leerheitsproblem**)

# Abhängigkeits-Graph

Idee für Erreichbarkeit: **Abhängigkeits-Graph**

... hier:



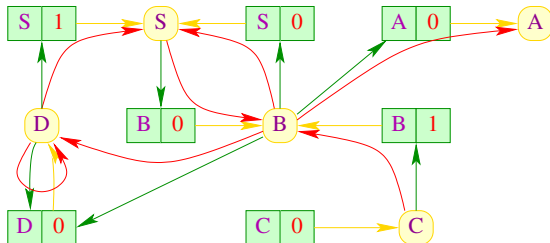
**Knoten:** Nichtterminale

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

# Abhängigkeits-Graph

Idee für Erreichbarkeit: **Abhängigkeits-Graph**

... hier:



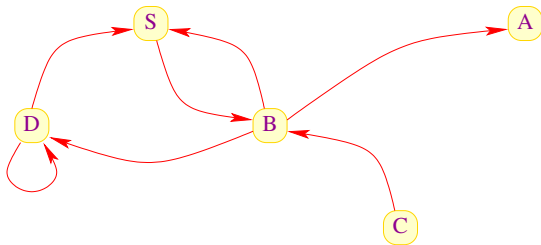
**Knoten:** Nichtterminale

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

# Abhängigkeits-Graph

Idee für Erreichbarkeit: **Abhängigkeits-Graph**

... hier:



**Knoten:** Nichtterminale

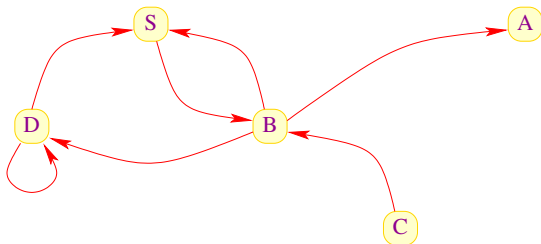
**Kanten:**  $(A, B)$  falls  $B \rightarrow \alpha_1 A \alpha_2 \in P$



# Abhängigkeits-Graph

Idee für Erreichbarkeit: **Abhängigkeits-Graph**

... hier:

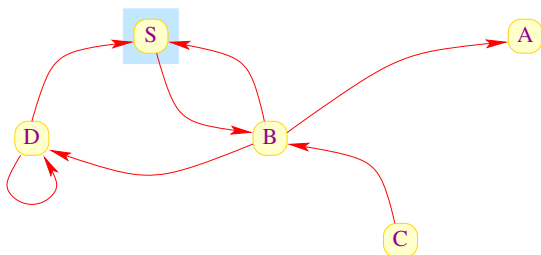


Das Nichtterminal **A** ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von **A** nach **S** gibt

# Abhängigkeits-Graph

Idee für Erreichbarkeit: **Abhängigkeits-Graph**

... hier:

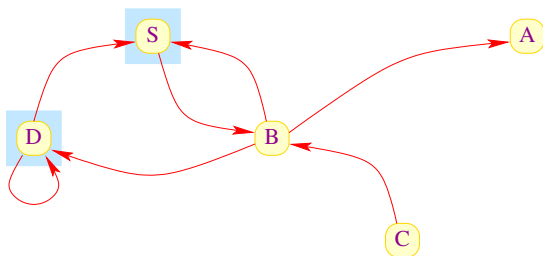


Das Nichtterminal **A** ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von **A** nach **S** gibt

# Abhängigkeits-Graph

Idee für Erreichbarkeit: **Abhängigkeits-Graph**

... hier:

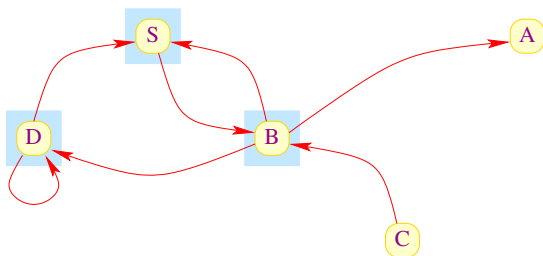


Das Nichtterminal **A** ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von **A** nach **S** gibt

# Abhängigkeits-Graph

Idee für Erreichbarkeit: **Abhängigkeits-Graph**

... hier:

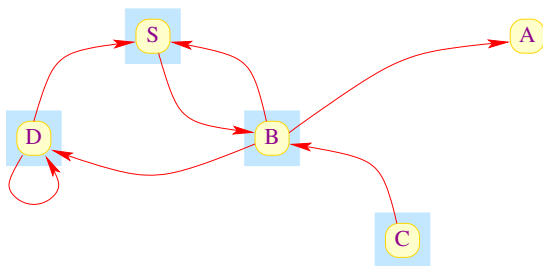


Das Nichtterminal **A** ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von **A** nach **S** gibt

# Abhängigkeits-Graph

Idee für Erreichbarkeit: **Abhängigkeits-Graph**

... hier:



Das Nichtterminal **A** ist erreichbar, falls es im Abhängigkeitsgraphen einen Pfad von **A** nach **S** gibt

## Fazit:

- Erreichbarkeit in gerichteten Graphen kann mithilfe von DFS in **linearer Zeit** berechnet werden.
- Damit kann die Menge aller erreichbaren und produktiven Nichtterminale in **linearer Zeit** berechnet werden

## Fazit:

- Erreichbarkeit in gerichteten Graphen kann mithilfe von DFS in **linearer Zeit** berechnet werden.
- Damit kann die Menge aller erreichbaren und produktiven Nichtterminale in **linearer Zeit** berechnet werden

Eine Grammatik  $G$  heißt **reduziert**, wenn alle Nichtterminale von  $G$  sowohl produktiv wie erreichbar sind ...

## Fazit:

- Erreichbarkeit in gerichteten Graphen kann mithilfe von DFS in **linearer Zeit** berechnet werden.
- Damit kann die Menge aller erreichbaren und produktiven Nichtterminale in **linearer Zeit** berechnet werden

Eine Grammatik  $G$  heißt **reduziert**, wenn alle Nichtterminale von  $G$  sowohl produktiv wie erreichbar sind ...

### Satz:

Zu jeder kontextfreien Grammatik  $G = (N, T, P, S)$  mit  $\mathcal{L}(G) \neq \emptyset$  kann in **linearer Zeit** eine reduzierte Grammatik  $G'$  konstruiert werden mit

$$\mathcal{L}(G) = \mathcal{L}(G')$$



## Konstruktion:

### 1. Schritt:

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. Schritt:

Konstruiere: 
$$P_1 = \{A \rightarrow \alpha \in P \mid A \in N_1 \wedge \alpha \in (N_1 \cup T)^*\}$$

## Konstruktion:

### 1. Schritt:

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. Schritt:

Konstruiere: 
$$P_1 = \{A \rightarrow \alpha \in P \mid A \in N_1 \wedge \alpha \in (N_1 \cup T)^*\}$$

### 3. Schritt:

Berechne die Teilmenge  $N_2 \subseteq N_1$  aller produktiven **und** erreichbaren Nichtterminale von  $G$ .

Da  $\mathcal{L}(G) \neq \emptyset$  ist insbesondere  $S \in N_2$ .

### 4. Schritt:

Konstruiere: 
$$P_2 = \{A \rightarrow \alpha \in P \mid A \in N_2 \wedge \alpha \in (N_2 \cup T)^*\}$$

## Konstruktion:

### 1. Schritt:

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. Schritt:

Konstruiere: 
$$P_1 = \{A \rightarrow \alpha \in P \mid A \in N_1 \wedge \alpha \in (N_1 \cup T)^*\}$$

### 3. Schritt:

Berechne die Teilmenge  $N_2 \subseteq N_1$  aller produktiven **und** erreichbaren Nichtterminale von  $G$ .

Da  $\mathcal{L}(G) \neq \emptyset$  ist insbesondere  $S \in N_2$ .

### 4. Schritt:

Konstruiere: 
$$P_2 = \{A \rightarrow \alpha \in P \mid A \in N_2 \wedge \alpha \in (N_2 \cup T)^*\}$$

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

... im Beispiel:

$$\begin{array}{l} S \rightarrow aBB \mid bD \\ A \rightarrow Bc \\ B \rightarrow Sd \mid C \\ C \rightarrow a \\ D \rightarrow BD \end{array}$$

... im Beispiel:

$$\begin{array}{l} S \rightarrow aBB \mid bD \\ A \rightarrow Bc \\ B \rightarrow Sd \mid C \\ C \rightarrow a \\ D \rightarrow BD \end{array}$$

... im Beispiel:

$$\begin{array}{l} S \rightarrow aBB \\ A \rightarrow Bc \\ B \rightarrow Sd \quad | \quad C \\ C \rightarrow a \end{array}$$

... im Beispiel:

$$\begin{array}{l} S \rightarrow aBB \\ A \rightarrow Bc \\ B \rightarrow Sd \quad | \quad C \\ C \rightarrow a \end{array}$$

... im Beispiel:

$S \rightarrow aBB$

$B \rightarrow Sd \mid C$

$C \rightarrow a$

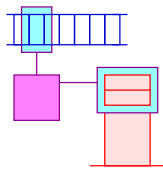


# **Kapitel 3:**

## **Grundlagen: Kellerautomaten**

# Grundlagen: Kellerautomaten

Durch kontextfreie Grammatiken spezifizierte Sprachen können durch Kellerautomaten (**Pushdown Automata**) akzeptiert werden:



Der Keller wird z.B. benötigt, um korrekte Klammerung zu überprüfen

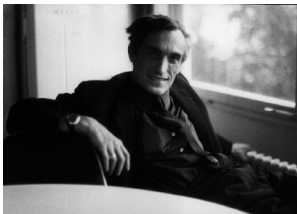


Friedrich L. Bauer, TUM



Klaus Samelson, TUM

Kellerautomaten für kontextfreie Sprachen wurden erstmals vorgeschlagen von Marcel-Paul Schützenberger und Antony G. Öttinger:



Marcel-Paul Schützenberger  
(1920-1996), Paris



Antony G. Öttinger, Präsident  
der ACM 1966-68

## Beispiel:

**Zustände:** 0, 1, 2

**Anfangszustand:** 0

**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

## Beispiel:

**Zustände:** 0, 1, 2  
**Anfangszustand:** 0  
**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

## Achtung:

- Wir unterscheiden **nicht** zwischen Kellersymbolen und Zuständen
- Das rechteste / oberste Kellersymbol repräsentiert den Zustand
- Jeder Übergang liest / modifiziert einen oberen Abschnitt des Kellers

# Kellerautomaten

## Definition:

Formal definieren wir deshalb einen **Kellerautomaten (PDA)** als ein Tupel:  $M = (Q, T, \delta, q_0, F)$  mit:

- $Q$  eine endliche Menge von Zuständen;
- $T$  das Eingabe-Alphabet;
- $q_0 \in Q$  der Anfangszustand;
- $F \subseteq Q$  die Menge der Endzustände und
- $\delta \subseteq Q^+ \times (T \cup \{\epsilon\}) \times Q^*$  eine endliche Menge von Übergängen

# Kellerautomaten

## Definition:

Formal definieren wir deshalb einen **Kellerautomaten (PDA)** als ein Tupel:  $M = (Q, T, \delta, q_0, F)$  mit:

- $Q$  eine endliche Menge von Zuständen;
- $T$  das Eingabe-Alphabet;
- $q_0 \in Q$  der Anfangszustand;
- $F \subseteq Q$  die Menge der Endzustände und
- $\delta \subseteq Q^+ \times (T \cup \{\epsilon\}) \times Q^*$  eine endliche Menge von Übergängen

Mithilfe der Übergänge definieren wir **Berechnungen** von Kellerautomaten

Der jeweilige **Berechnungszustand** (die aktuelle **Konfiguration**) ist ein Paar:

$$(\gamma, w) \in Q^* \times T^*$$

bestehend aus dem **Kellerinhalt** und dem **noch zu lesenden Input**.



## ... im Beispiel:

**Zustände:** 0, 1, 2

**Anfangszustand:** 0

**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

## ... im Beispiel:

**Zustände:** 0, 1, 2

**Anfangszustand:** 0

**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

(0, *aaabbb*)

## ... im Beispiel:

**Zustände:** 0, 1, 2

**Anfangszustand:** 0

**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

$(0, \textit{aaabbb}) \vdash (11, \textit{aabbb})$

## ... im Beispiel:

**Zustände:** 0, 1, 2

**Anfangszustand:** 0

**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

$(0, \textit{aaabbb}) \vdash (11, \textit{aabbb})$   
 $\vdash (111, \textit{abbb})$

## ... im Beispiel:

**Zustände:** 0, 1, 2

**Anfangszustand:** 0

**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

$(0, \textit{aaabbb}) \vdash (11, \textit{aabbb})$   
 $\vdash (111, \textit{abb})$   
 $\vdash (1111, \textit{bbb})$

## ... im Beispiel:

**Zustände:** 0, 1, 2

**Anfangszustand:** 0

**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

$(0, \textit{aaabbb}) \vdash (11, \textit{aaabbb})$   
 $\vdash (111, \textit{abbb})$   
 $\vdash (1111, \textit{bbb})$   
 $\vdash (112, \textit{bb})$

## ... im Beispiel:

**Zustände:** 0, 1, 2

**Anfangszustand:** 0

**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

$(0, \text{aaabbb}) \vdash (11, \text{aabbb})$   
 $\vdash (111, \text{abbb})$   
 $\vdash (1111, \text{bbb})$   
 $\vdash (112, \text{bb})$   
 $\vdash (12, \text{b})$

## ... im Beispiel:

**Zustände:** 0, 1, 2

**Anfangszustand:** 0

**Endzustände:** 0, 2

0	<i>a</i>	11
1	<i>a</i>	11
11	<i>b</i>	2
12	<i>b</i>	2

$(0, \textit{aaabbb}) \vdash (11, \textit{aaabbb})$   
 $\vdash (111, \textit{abbb})$   
 $\vdash (1111, \textit{bbb})$   
 $\vdash (112, \textit{bb})$   
 $\vdash (12, \textit{b})$   
 $\vdash (2, \epsilon)$



Ein Berechnungsschritt wird durch die Relation  $\vdash \subseteq (Q^* \times T^*)^2$  beschrieben, wobei

$$(\alpha\gamma, xw) \vdash (\alpha\gamma', w) \quad \text{für} \quad (\gamma, x, \gamma') \in \delta$$

Ein Berechnungsschritt wird durch die Relation  $\vdash \subseteq (Q^* \times T^*)^2$  beschrieben, wobei

$$(\alpha\gamma, xw) \vdash (\alpha\gamma', w) \quad \text{für} \quad (\gamma, x, \gamma') \in \delta$$

## Bemerkungen:

- Die Relation  $\vdash$  hängt natürlich vom Kellerautomaten  $M$  ab
- Die reflexive und transitive Hülle von  $\vdash$  bezeichnen wir mit  $\vdash^*$
- Dann ist die von  $M$  akzeptierte Sprache:

$$\mathcal{L}(M) = \{w \in T^* \mid \exists f \in F : (q_0, w) \vdash^* (f, \epsilon)\}$$

Ein Berechnungsschritt wird durch die Relation  $\vdash \subseteq (Q^* \times T^*)^2$  beschrieben, wobei

$$(\alpha\gamma, xw) \vdash (\alpha\gamma', w) \quad \text{für} \quad (\gamma, x, \gamma') \in \delta$$

## Bemerkungen:

- Die Relation  $\vdash$  hängt natürlich vom Kellerautomaten  $M$  ab
- Die reflexive und transitive Hülle von  $\vdash$  bezeichnen wir mit  $\vdash^*$
- Dann ist die von  $M$  akzeptierte Sprache:

$$\mathcal{L}(M) = \{w \in T^* \mid \exists f \in F : (q_0, w) \vdash^* (f, \epsilon)\}$$

Wir akzeptieren also mit **Endzustand** und leerem Keller

# Deterministischer Kellerautomat

## Definition:

Der Kellerautomat  $M$  heißt **deterministisch**, falls für alle verschiedenen Übergänge  $(\gamma_1, x, \gamma_2), (\gamma'_1, x', \gamma'_2) \in \delta$  gilt:  
Ist  $\gamma_1$  ein Suffix von  $\gamma'_1$ , dann muss  $x \neq x' \wedge x \neq \epsilon \neq x'$  sein.

Dies stellt sicher, dass jede Konfiguration maximal eine Nachfolge-Konfiguration hat.

# Deterministischer Kellerautomat

## Definition:

Der Kellerautomat  $M$  heißt **deterministisch**, falls für alle verschiedenen Übergänge  $(\gamma_1, x, \gamma_2), (\gamma'_1, x', \gamma'_2) \in \delta$  gilt:  
Ist  $\gamma_1$  ein Suffix von  $\gamma'_1$ , dann muss  $x \neq x' \wedge x \neq \epsilon \neq x'$  sein.

Dies stellt sicher, dass jede Konfiguration maximal eine Nachfolge-Konfiguration hat.

... im Beispiel:

0	$a$	11
1	$a$	11
11	$b$	2
12	$b$	2

... ist das natürlich der Fall

# Shift-Reduce-Parser

## Satz:

Zu jeder kontextfreien Grammatik  $G = (N, T, P, S)$  kann ein Kellerautomat  $M$  konstruiert werden mit  $\mathcal{L}(G) = \mathcal{L}(M)$ .

Der Satz ist für uns so wichtig, dass wir **zwei** Konstruktionen angeben, resultierend in den Automaten  $M_G^{(1)}$  und  $M_G^{(2)}$ :

# Shift-Reduce-Parser

## Satz:

Zu jeder kontextfreien Grammatik  $G = (N, T, P, S)$  kann ein Kellerautomat  $M$  konstruiert werden mit  $\mathcal{L}(G) = \mathcal{L}(M)$ .

Der Satz ist für uns so wichtig, dass wir **zwei** Konstruktionen angeben, resultierend in den Automaten  $M_G^{(1)}$  und  $M_G^{(2)}$ :

## Konstruktion 1: Shift-Reduce-Parser

- Die Eingabe wird sukzessive auf den Keller geschoben.
- Liegt oben auf dem Keller eine **vollständige rechte Seite** (ein **Handle**) vor, wird dieses durch die zugehörige linke Seite ersetzt (**reduziert**)

# Shift-Reduce-Parser

Beispiel:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

Der Kellerautomat:

**Zustände:**  $q_0, f, a, b, A, B, S$ ;  
**Anfangszustand:**  $q_0$   
**Endzustand:**  $f$

$q_0$	$a$	$q_0 a$
$a$	$\epsilon$	$A$
$A$	$b$	$Ab$
$b$	$\epsilon$	$B$
$AB$	$\epsilon$	$S$
$q_0 S$	$\epsilon$	$f$



# Shift-Reduce-Parser

**Konstruktion:** Allgemein konstruieren wir einen Automaten

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

- $Q = T \cup N \cup \{q_0, f\}$  ( $q_0, f$  neu);
- $F = \{f\}$ ;
- Übergänge:

$$\begin{aligned} \delta = & \{(q, x, qx) \mid q \in Q, x \in T\} \cup // \text{ Shift-Übergänge} \\ & \{(q \alpha, \epsilon, qA) \mid q \in Q, A \rightarrow \alpha \in P\} \cup // \text{ Reduce-Übergänge} \\ & \{(q_0 S, \epsilon, f)\} // \text{ Abschluss} \end{aligned}$$

# Shift-Reduce-Parser

**Konstruktion:** Allgemein konstruieren wir einen Automaten

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

- $Q = T \cup N \cup \{q_0, f\}$  ( $q_0, f$  neu);
- $F = \{f\}$ ;
- Übergänge:

$$\begin{aligned} \delta = & \{(q, x, qx) \mid q \in Q, x \in T\} \cup // \text{ Shift-Übergänge} \\ & \{(q\alpha, \epsilon, qA) \mid q \in Q, A \rightarrow \alpha \in P\} \cup // \text{ Reduce-Übergänge} \\ & \{(q_0 S, \epsilon, f)\} // \text{ Abschluss} \end{aligned}$$

**Beispiel-Berechnung:**

$$\begin{array}{l} (q_0, ab) \vdash (q_0 a, b) \vdash (q_0 A, b) \\ \vdash (q_0 A b, \epsilon) \vdash (q_0 AB, \epsilon) \\ \vdash (q_0 S, \epsilon) \vdash (f, \epsilon) \end{array}$$

# Shift-Reduce-Parser

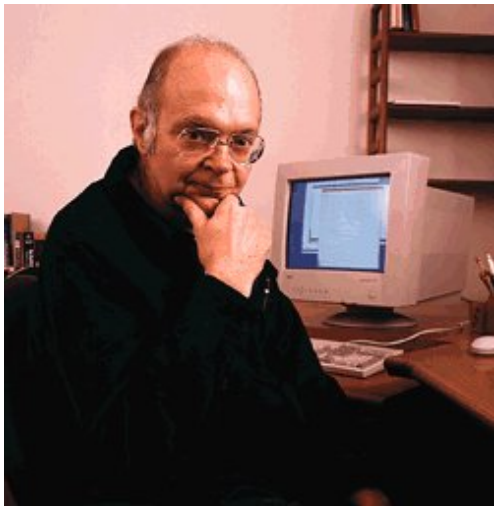
## Offenbar gilt:

- Die Folge der Reduktionen entspricht einer **reversen Rechtsableitung** für die Eingabe
- Zur Korrektheit zeigt man, dass gilt:

$$(q, w) \vdash^* (qA, \epsilon) \quad \text{gdw.} \quad A \rightarrow^* w$$

- Der Shift-Reduce-Kellerautomat  $M_G^{(1)}$  ist i.a. **nicht-deterministisch**
- Um ein deterministisches Parse-Verfahren zu erhalten, muss man die Reduktionsstellen identifizieren

$\implies$  LR-Parsing



Donald E. Knuth, Stanford

# Item-Kellerautomat

## Konstruktion 2: Item-Kellerautomat

- Rekonstruiere eine **Linksableitung**.
- Expandiere Nichtterminale mithilfe einer Regel.
- Verifiziere sukzessive, dass die gewählte Regel mit der Eingabe übereinstimmt.  
⇒ Die Zustände sind jetzt **Items**.
- Ein Item ist eine Regel mit **Punkt**:

$$[A \rightarrow \alpha \bullet \beta], \quad A \rightarrow \alpha \beta \in P$$

Der Punkt gibt an, wieweit die Regel bereits abgearbeitet wurde

# Item-Kellerautomat

Unser Beispiel:

$$S \rightarrow AB \quad A \rightarrow a \quad B \rightarrow b$$

Wir fügen eine Regel:  $S' \rightarrow S$  hinzu

Dann konstruieren wir:

**Anfangszustand:**  $[S' \rightarrow \bullet S]$

**Endzustand:**  $[S' \rightarrow S \bullet]$

$[S' \rightarrow \bullet S]$	$\epsilon$	$[S' \rightarrow \bullet S] [S \rightarrow \bullet AB]$
$[S \rightarrow \bullet AB]$	$\epsilon$	$[S \rightarrow \bullet AB] [A \rightarrow \bullet a]$
$[A \rightarrow \bullet a]$	$a$	$[A \rightarrow a \bullet]$
$[S \rightarrow \bullet AB] [A \rightarrow a \bullet]$	$\epsilon$	$[S \rightarrow A \bullet B]$
$[S \rightarrow A \bullet B]$	$\epsilon$	$[S \rightarrow A \bullet B] [B \rightarrow \bullet b]$
$[B \rightarrow \bullet b]$	$b$	$[B \rightarrow b \bullet]$
$[S \rightarrow A \bullet B] [B \rightarrow b \bullet]$	$\epsilon$	$[S \rightarrow AB \bullet]$
$[S' \rightarrow \bullet S] [S \rightarrow AB \bullet]$	$\epsilon$	$[S' \rightarrow S \bullet]$

# Item-Kellerautomat

Der Item-Kellerautomat  $M_G^{(2)}$  hat drei Arten von Übergängen:

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

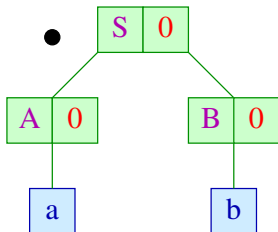
**Shifts:**  $([A \rightarrow \alpha \bullet a \beta], a, [A \rightarrow \alpha a \bullet \beta])$  für  $A \rightarrow \alpha a \beta \in P$

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

Items der Form:  $[A \rightarrow \alpha \bullet]$  heißen auch **vollständig**  
Der Item-Kellerautomat schiebt den Punkt einmal um den  
Ableitungsbaum herum ...

# Item-Kellerautomat

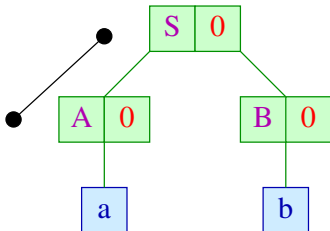
... im Beispiel:





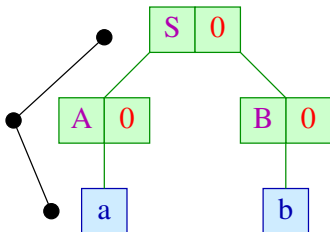
# Item-Kellerautomat

... im Beispiel:



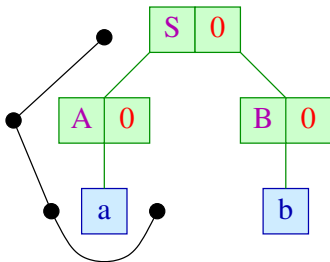
# Item-Kellerautomat

... im Beispiel:



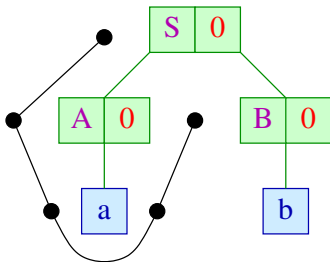
# Item-Kellerautomat

... im Beispiel:



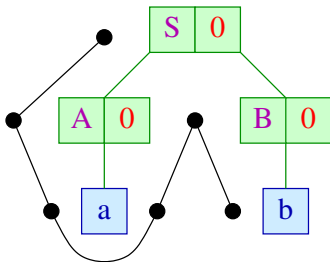
# Item-Kellerautomat

... im Beispiel:



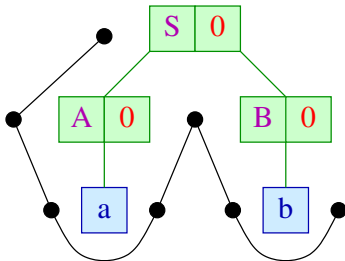
# Item-Kellerautomat

... im Beispiel:



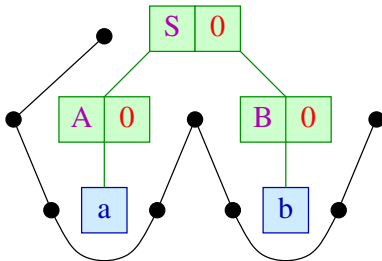
# Item-Kellerautomat

... im Beispiel:



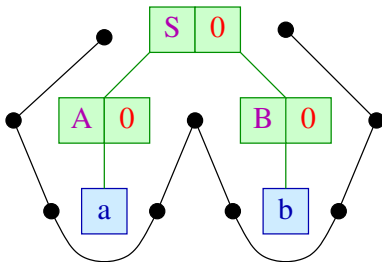
# Item-Kellerautomat

... im Beispiel:



# Item-Kellerautomat

... im Beispiel:





# Item-Kellerautomat

## Diskussion:

- Die **Expansionen** einer Berechnung bilden eine **Linksableitung**
- Leider muss man bei den Expansionen **nichtdeterministisch** zwischen verschiedenen Regeln auswählen
- Zur Korrektheit der Konstruktion zeigt man, dass für jedes Item  $[A \rightarrow \alpha \bullet B \beta]$  gilt:

$$([A \rightarrow \alpha \bullet B \beta], w) \vdash^* ([A \rightarrow \alpha B \bullet \beta], \epsilon) \quad \text{gdw.} \quad B \xrightarrow{*} w$$

- **LL-Parsing** basiert auf dem Item-Kellerautomaten und versucht, die Expansionen durch **Vorausschau** deterministisch zu machen

...



Philip M. Lewis, SUNY



Richard E. Stearns, SUNY

Beispiel:  $S \rightarrow \epsilon \mid aSb$

Die Übergänge des zugehörigen Item-Kellerautomat:

0	$[S' \rightarrow \bullet S]$	$\epsilon$	$[S' \rightarrow \bullet S] [S \rightarrow \bullet]$
1	$[S' \rightarrow \bullet S]$	$\epsilon$	$[S' \rightarrow \bullet S] [S \rightarrow \bullet aSb]$
2	$[S \rightarrow \bullet aSb]$	$a$	$[S \rightarrow a \bullet Sb]$
3	$[S \rightarrow a \bullet Sb]$	$\epsilon$	$[S \rightarrow a \bullet Sb] [S \rightarrow \bullet]$
4	$[S \rightarrow a \bullet Sb]$	$\epsilon$	$[S \rightarrow a \bullet Sb] [S \rightarrow \bullet aSb]$
5	$[S \rightarrow a \bullet Sb] [S \rightarrow \bullet]$	$\epsilon$	$[S \rightarrow aS \bullet b]$
6	$[S \rightarrow a \bullet Sb] [S \rightarrow aSb \bullet]$	$\epsilon$	$[S \rightarrow aS \bullet b]$
7	$[S \rightarrow aS \bullet b]$	$b$	$[S \rightarrow aSb \bullet]$
8	$[S' \rightarrow \bullet S] [S \rightarrow \bullet]$	$\epsilon$	$[S' \rightarrow S \bullet]$
9	$[S' \rightarrow \bullet S] [S \rightarrow aSb \bullet]$	$\epsilon$	$[S' \rightarrow S \bullet]$

Konflikte gibt es zwischen den Übergängen (0,1) bzw. zwischen (3,4) – die sich durch Betrachten des nächsten Zeichens lösen ließen

# **Kapitel 4:**

## **Vorausschau-Mengen**

# Vorausschau-Mengen

## Definition:

Für eine Menge  $L \subseteq T^*$  definieren wir:

$$\text{First}_k(L) = \{u \in L \mid |u| < k\} \cup \{u \in T^k \mid \exists v \in T^* : uv \in L\}$$

Beispiel:

$\epsilon$
$ab$
$aabb$
$aaabbb$

# Vorausschau-Mengen

## Definition:

Für eine Menge  $L \subseteq T^*$  definieren wir:

$$\text{First}_k(L) = \{u \in L \mid |u| < k\} \cup \{u \in T^k \mid \exists v \in T^* : uv \in L\}$$

Beispiel:

$\epsilon$
$ab$
$aabb$
$aaabbb$

die Präfixe der Länge 2

# Vorausschau-Mengen

## Rechenregeln:

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

$$\begin{aligned}\text{First}_k(\emptyset) &= \emptyset \\ \text{First}_k(L_1 \cup L_2) &= \text{First}_k(L_1) \cup \text{First}_k(L_2) \\ \text{First}_k(L_1 \cdot L_2) &= \text{First}_k(\text{First}_k(L_1) \cdot \text{First}_k(L_2)) \\ &:= \text{First}_k(L_1) \odot \text{First}_k(L_2)\end{aligned}$$

## Beachte:

- Die Menge  $\mathbb{D}_k = 2^{T^{\leq k}}$  ist **endlich**
- Die Operation:  $\odot : \mathbb{D}_k \times \mathbb{D}_k \rightarrow \mathbb{D}_k$  ist **distributiv** in jedem Argument:

$$\begin{aligned}L \odot \emptyset &= \emptyset & L \odot (L_1 \cup L_2) &= (L \odot L_1) \cup (L \odot L_2) \\ \emptyset \odot L &= \emptyset & (L_1 \cup L_2) \odot L &= (L_1 \odot L) \cup (L_2 \odot L)\end{aligned}$$

# First<sub>k</sub>

Für  $\alpha \in (N \cup T)^*$  sind wir interessiert an der Menge:

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

Für  $k \geq 1$  gilt:

$$\begin{aligned} \text{First}_k(x) &= \{x\} && \text{für } x \in T \cup \{\epsilon\} \\ \text{First}_k(\alpha_1 \alpha_2) &= \text{First}_k(\alpha_1) \odot \text{First}_k(\alpha_2) \end{aligned}$$



# First<sub>k</sub>

Für  $\alpha \in (N \cup T)^*$  sind wir interessiert an der Menge:

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

Für  $k \geq 1$  gilt:

$$\begin{aligned} \text{First}_k(x) &= \{x\} && \text{für } x \in T \cup \{\epsilon\} \\ \text{First}_k(\alpha_1 \alpha_2) &= \text{First}_k(\alpha_1) \odot \text{First}_k(\alpha_2) \end{aligned}$$

**Frage:** Wie berechnet man  $\text{First}_k(A)$  ??

# First<sub>k</sub>

Für  $\alpha \in (N \cup T)^*$  sind wir interessiert an der Menge:

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

Für  $k \geq 1$  gilt:

$$\begin{aligned} \text{First}_k(x) &= \{x\} && \text{für } x \in T \cup \{\epsilon\} \\ \text{First}_k(\alpha_1 \alpha_2) &= \text{First}_k(\alpha_1) \odot \text{First}_k(\alpha_2) \end{aligned}$$

**Frage:** Wie berechnet man  $\text{First}_k(A)$  ??

**Idee:** Stelle ein **Ungleichungssystem** auf!

## First<sub>2</sub>

Beispiel:  $k = 2$

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

Jede Regel gibt Anlass zu einer Inklusionsbeziehung:

$\text{First}_2(E) \supseteq \text{First}_2(E + T)$

$\text{First}_2(T) \supseteq \text{First}_2(T * F)$

$\text{First}_2(F) \supseteq \text{First}_2(( E ))$

$\text{First}_2(E) \supseteq \text{First}_2(T)$

$\text{First}_2(T) \supseteq \text{First}_2(F)$

$\text{First}_2(F) \supseteq \{\text{name}, \text{int}\}$

# First<sub>2</sub>

Beispiel:  $k = 2$

$$\begin{array}{lcl} E & \rightarrow & E + T \quad | \quad T \\ T & \rightarrow & T * F \quad | \quad F \\ F & \rightarrow & ( E ) \quad | \quad \text{name} \quad | \quad \text{int} \end{array}$$

Jede Regel gibt Anlass zu einer Inklusionsbeziehung:

$$\begin{array}{ll} \text{First}_2(E) \supseteq \text{First}_2(E + T) & \text{First}_2(E) \supseteq \text{First}_2(T) \\ \text{First}_2(T) \supseteq \text{First}_2(T * F) & \text{First}_2(T) \supseteq \text{First}_2(F) \\ \text{First}_2(F) \supseteq \text{First}_2(( E )) & \text{First}_2(F) \supseteq \{\text{name}, \text{int}\} \end{array}$$

Eine Inklusion  $\text{First}_2(E) \supseteq \text{First}_2(E + T)$  kann weiter vereinfacht werden zu:

$$\text{First}_2(E) \supseteq \text{First}_2(E) \odot \{+\} \odot \text{First}_2(T)$$

## Ungleichungssystem für $\text{First}_2$

Insgesamt erhalten wir das Ungleichungssystem:

$$\begin{array}{ll} \text{First}_2(E) \supseteq \text{First}_2(E) \odot \{+\} \odot \text{First}_2(T) & \text{First}_2(E) \supseteq \text{First}_2(T) \\ \text{First}_2(T) \supseteq \text{First}_2(T) \odot \{*\} \odot \text{First}_2(F) & \text{First}_2(T) \supseteq \text{First}_2(F) \\ \text{First}_2(F) \supseteq \{(\} \odot \text{First}_2(E) \odot \{)\} & \text{First}_2(F) \supseteq \{\text{name, int}\} \end{array}$$

# Ungleichungssystem für $\text{First}_k$

Insgesamt erhalten wir das Ungleichungssystem:

$$\begin{array}{ll} \text{First}_2(E) \supseteq \text{First}_2(E) \odot \{+\} \odot \text{First}_2(T) & \text{First}_2(E) \supseteq \text{First}_2(T) \\ \text{First}_2(T) \supseteq \text{First}_2(T) \odot \{*\} \odot \text{First}_2(F) & \text{First}_2(T) \supseteq \text{First}_2(F) \\ \text{First}_2(F) \supseteq \{(\} \odot \text{First}_2(E) \odot \{)\} & \text{First}_2(F) \supseteq \{\text{name, int}\} \end{array}$$

Allgemein:

$$\text{First}_k(A) \supseteq \text{First}_k(X_1) \odot \dots \odot \text{First}_k(X_m)$$

für jede Regel  $A \rightarrow X_1 \dots X_m \in P$  mit  $X_i \in T \cup N$ .

# Ungleichungssystem für First<sub>2</sub>

## Gesucht:

- möglichst **kleine** Lösung
- Algorithmus, der diese berechnet

## ... im Beispiel:

$$\begin{array}{lcl} \text{First}_2(E) \supseteq & \text{First}_2(E) \odot \{+\} \odot \text{First}_2(T) & \text{First}_2(E) \supseteq \text{First}_2(T) \\ \text{First}_2(T) \supseteq & \text{First}_2(T) \odot \{*\} \odot \text{First}_2(F) & \text{First}_2(T) \supseteq \text{First}_2(F) \\ \text{First}_2(F) \supseteq & \{(\} \odot \text{First}_2(E) \odot \{)\} & \text{First}_2(F) \supseteq \{\text{name, int}\} \end{array}$$

... hat die Lösung:

<i>E</i>	name, int, (name, (int, ((, name *, int *, name +, int +
<i>T</i>	name, int, (name, (int, ((, name *, int *
<i>F</i>	name, int, (name, (int, ((

# Ungleichungssystem für $\text{First}_k$

## Beobachtung:

- Die Menge  $\mathbb{D}_k$  der möglichen Werte für  $\text{First}_k(A)$  bilden einen **vollständigen Verband**
- Die Operatoren auf den rechten Seiten der Ungleichungen sind **monoton**, d.h. verträglich mit  $\subseteq$



# **Kapitel 5:**

## **Exkurs: Vollständige Verbände**

# Verbände

## Definition:

Eine Menge  $\mathbb{D}$  mit einer Relation  $\sqsubseteq \subseteq \mathbb{D} \times \mathbb{D}$  ist ein **Verband (Lattice)** falls für alle  $a, b, c \in \mathbb{D}$  gilt:

$$a \sqsubseteq a$$

*Reflexivität*

$$a \sqsubseteq b \wedge b \sqsubseteq a \implies a = b$$

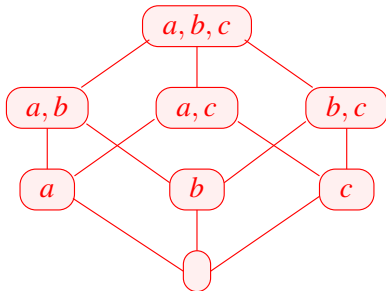
*Anti – Symmetrie*

$$a \sqsubseteq b \wedge b \sqsubseteq c \implies a \sqsubseteq c$$

*Transitivität*

## Beispiele:

1.  $\mathbb{D} = 2^{\{a,b,c\}}$  mit der Relation " $\subseteq$ ":

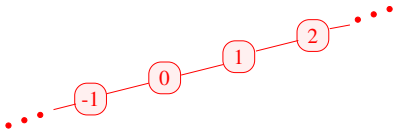


## Verbände (Beispiele)

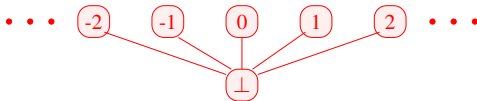
3.  $\mathbb{Z}$  mit der Relation “=” :



3.  $\mathbb{Z}$  mit der Relation “ $\leq$ ” :



4.  $\mathbb{Z}_{\perp} = \mathbb{Z} \cup \{\perp\}$  mit der Ordnung:



# Obere Schranken

## Definition:

$d \in \mathbb{D}$  heißt **obere Schranke** für  $X \subseteq \mathbb{D}$  falls

$$x \sqsubseteq d \quad \text{für alle } x \in X$$

# Obere Schranken

## Definition:

$d \in \mathbb{D}$  heißt **obere Schranke** für  $X \subseteq \mathbb{D}$  falls

$$x \sqsubseteq d \quad \text{für alle } x \in X$$

## Definition:

$d$  heißt **kleinste obere Schranke (lub)** falls

- 1  $d$  eine obere Schranke ist und
- 2  $d \sqsubseteq y$  für jede obere Schranke  $y$  für  $X$ .

# Obere Schranken

## Definition:

$d \in \mathbb{D}$  heißt **obere Schranke** für  $X \subseteq \mathbb{D}$  falls

$$x \leq d \quad \text{für alle } x \in X$$

## Definition:

$d$  heißt **kleinste obere Schranke (lub)** falls

- 1  $d$  eine obere Schranke ist und
- 2  $d \leq y$  für jede obere Schranke  $y$  für  $X$ .

## Achtung:

- $\{0, 2, 4, \dots\} \subseteq \mathbb{Z}$  besitzt **keine** obere Schranke!
- $\{0, 2, 4\} \subseteq \mathbb{Z}$  besitzt die oberen Schranken **4, 5, 6, ...**

# Vollständige Verbände

## Definition:

Ein **vollständiger Verband** (cl)  $\mathbb{D}$  ist eine Halbordnung, in der **jede Teilmenge**  $X \subseteq \mathbb{D}$  eine kleinste obere Schranke  $\bigsqcup X \in \mathbb{D}$  besitzt.

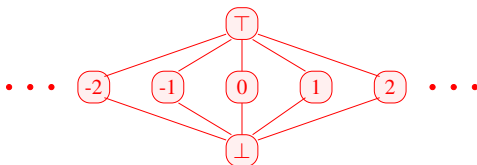
## Beachte:

Jeder vollständige Verband besitzt

- ein **kleinstes** Element  $\perp = \bigsqcup \emptyset \in \mathbb{D}$ ;
- ein **größtes** Element  $\top = \bigsqcup \mathbb{D} \in \mathbb{D}$ .

## Vollständige Verbände (Beispiele:)

- 1  $\mathbb{D} = 2^{\{a,b,c\}}$  ist ein cl
- 2  $\mathbb{D} = \mathbb{Z}$  mit “=” ist keiner.
- 3  $\mathbb{D} = \mathbb{Z}$  mit “ $\leq$ ” ebenfalls nicht.
- 4  $\mathbb{D} = \mathbb{Z}_{\perp}$  auch nicht
- 5 Mit einem zusätzlichen Symbol  $\top$  erhalten wir den **flachen** Verband  $\mathbb{Z}_{\perp}^{\top} = \mathbb{Z} \cup \{\perp, \top\}$  :





# Untere Schranken

Es gilt:

## Satz:

In jedem vollständigen Verband  $\mathbb{D}$  besitzt jede Teilmenge  $X \subseteq \mathbb{D}$  eine größte untere Schranke  $\bigwedge X$ .

# Untere Schranken

Es gilt:

## Satz:

In jedem vollständigen Verband  $\mathbb{D}$  besitzt jede Teilmenge  $X \subseteq \mathbb{D}$  eine größte untere Schranke  $\bigsqcap X$ .

## Beweis

**Konstruiere:**  $U = \{u \in \mathbb{D} \mid \forall x \in X : u \sqsubseteq x\}$ .  
// die Menge der unteren Schranken von  $X$

# Untere Schranken

Es gilt:

## Satz:

In jedem vollständigen Verband  $\mathbb{D}$  besitzt jede Teilmenge  $X \subseteq \mathbb{D}$  eine größte untere Schranke  $\bigwedge X$ .

## Beweis

**Konstruiere:**  $U = \{u \in \mathbb{D} \mid \forall x \in X : u \sqsubseteq x\}$ .

// die Menge der unteren Schranken von  $X$

**Setze:**  $g := \bigsqcup U$

**Behauptung:**  $g = \bigwedge X$

# Untere Schranken

## Beweis

**Konstruiere:**  $U = \{u \in \mathbb{D} \mid \forall x \in X : u \sqsubseteq x\}$ .  
// die Menge der unteren Schranken von  $X$

**Setze:**  $g := \bigsqcup U$

**Behauptung:**  $g = \bigsqcap X$

1  $g$  ist eine **untere Schranke** von  $X$ :

Für  $x \in X$  gilt:

$u \sqsubseteq x$  für alle  $u \in U$

$\implies x$  ist obere Schranke von  $U$

$\implies g \sqsubseteq x$

# Untere Schranken

## Beweis

**Konstruiere:**  $U = \{u \in \mathbb{D} \mid \forall x \in X : u \sqsubseteq x\}$ .  
// die Menge der unteren Schranken von  $X$

**Setze:**  $g := \bigsqcup U$

**Behauptung:**  $g = \bigsqcap X$

1  $g$  ist eine **untere Schranke** von  $X$ :

Für  $x \in X$  gilt:

$u \sqsubseteq x$  für alle  $u \in U$

$\implies x$  ist obere Schranke von  $U$

$\implies g \sqsubseteq x$

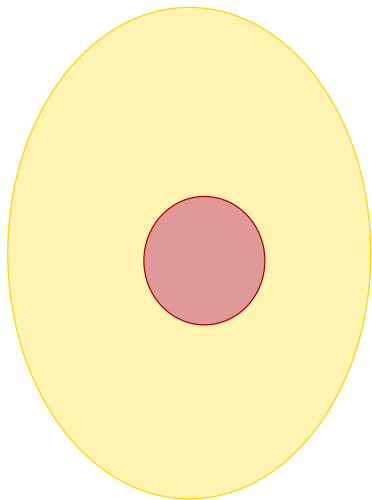
2  $g$  ist **größte untere Schranke** von  $X$ :

Für jede untere Schranke  $u$  von  $X$  gilt:

$u \in U$

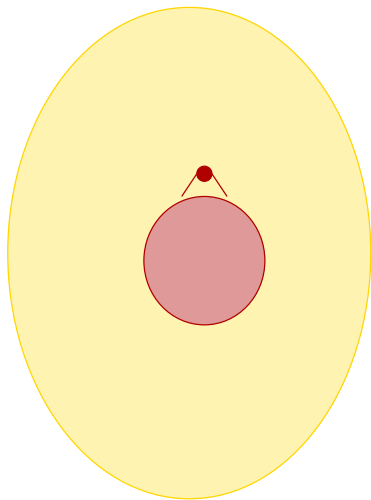
$\implies u \sqsubseteq g$

## Verbände – grafisch



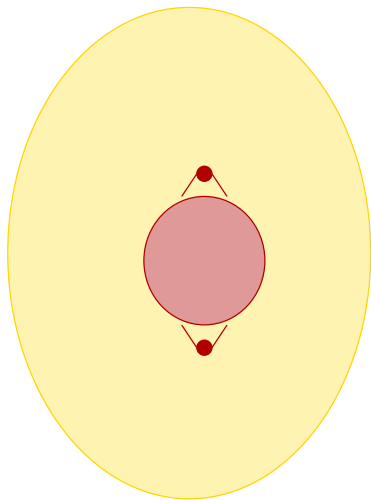
## Verbände – grafisch

- Kleinste obere Schranke



## Verbände – grafisch

- Kleinste obere Schranke
- Größte untere Schranke





# Ungleichungssysteme über Verbänden

## Problem:

Wir suchen **Lösungen** für Ungleichungssysteme der Form:

$$x_i \sqsupseteq f_i(x_1, \dots, x_n)$$

# Ungleichungssysteme über Verbänden

## Problem:

Wir suchen **Lösungen** für Ungleichungssysteme der Form:

$$x_i \sqsubseteq f_i(x_1, \dots, x_n)$$

wobei:

$x_i$	Unbekannte
$\mathbb{D}$	Werte
$\sqsubseteq \subseteq \mathbb{D} \times \mathbb{D}$	Ordnungsrelation
$f_i: \mathbb{D}^n \rightarrow \mathbb{D}$	Bedingung

# Ungleichungssysteme über Verbänden

## Problem:

Wir suchen **Lösungen** für Ungleichungssysteme der Form:

$$x_i \sqsupseteq f_i(x_1, \dots, x_n)$$

wobei:	$x_i$	Unbekannte	hier: $\text{First}_k(A)$
	$\mathbb{D}$	Werte	hier: $\mathbb{D}_k = 2^{T^{\leq k}}$
	$\sqsubseteq \subseteq \mathbb{D} \times \mathbb{D}$	Ordnungsrelation	hier: $\subseteq$
	$f_i: \mathbb{D}^n \rightarrow \mathbb{D}$	Bedingung	hier: ...

**im Beispiel: Ungleichung für  $\text{First}_k(A)$**

$$\text{First}_k(A) \sqsupseteq \bigcup \{ \text{First}_k(X_1) \odot \dots \odot \text{First}_k(X_m) \mid A \rightarrow X_1 \dots X_m \in P \}$$

# Ungleichungssysteme über Verbänden

## Problem:

Wir suchen **Lösungen** für Ungleichungssysteme der Form:

$$x_i \supseteq f_i(x_1, \dots, x_n)$$

wobei:	$x_i$	Unbekannte	hier: $\text{First}_k(A)$
	$\mathbb{D}$	Werte	hier: $\mathbb{D}_k = 2^{T^{\leq k}}$
	$\subseteq \subseteq \mathbb{D} \times \mathbb{D}$	Ordnungsrelation	hier: $\subseteq$
	$f_i: \mathbb{D}^n \rightarrow \mathbb{D}$	Bedingung	hier: ...

## im Beispiel: Ungleichung für $\text{First}_k(A)$

$$\text{First}_k(A) \supseteq \bigcup \{ \text{First}_k(X_1) \odot \dots \odot \text{First}_k(X_m) \mid A \rightarrow X_1 \dots X_m \in P \}$$

denn:  $x \supseteq d_1 \wedge \dots \wedge x \supseteq d_k$  gdw.  $x \supseteq \bigsqcup \{d_1, \dots, d_k\}$

# Monotonie

## Definition:

Eine Abbildung  $f : \mathbb{D}_1 \rightarrow \mathbb{D}_2$  heißt **monoton**, falls  $f(a) \sqsubseteq f(b)$  für alle  $a \sqsubseteq b$ .

# Monotonie

## Definition:

Eine Abbildung  $f : \mathbb{D}_1 \rightarrow \mathbb{D}_2$  heißt **monoton**, falls  $f(a) \sqsubseteq f(b)$  für alle  $a \sqsubseteq b$ .

## Beispiele:

- 1  $\mathbb{D}_1 = \mathbb{D}_2 = 2^U$  für eine Menge  $U$  und  $f x = (x \cap a) \cup b$ .  
Offensichtlich ist jedes solche  $f$  monoton

# Monotonie

## Definition:

Eine Abbildung  $f : \mathbb{D}_1 \rightarrow \mathbb{D}_2$  heißt **monoton**, falls  $f(a) \sqsubseteq f(b)$  für alle  $a \sqsubseteq b$ .

## Beispiele:

- 1  $\mathbb{D}_1 = \mathbb{D}_2 = 2^U$  für eine Menge  $U$  und  $f x = (x \cap a) \cup b$ .  
Offensichtlich ist jedes solche  $f$  monoton
- 2  $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{Z}$  (mit der Ordnung " $\leq$ "). Dann gilt:
  - $\text{inc } x = x + 1$  ist monoton.
  - $\text{dec } x = x - 1$  ist monoton.

# Monotonie

## Definition:

Eine Abbildung  $f : \mathbb{D}_1 \rightarrow \mathbb{D}_2$  heißt **monoton**, falls  $f(a) \sqsubseteq f(b)$  für alle  $a \sqsubseteq b$ .

## Beispiele:

- 1  $\mathbb{D}_1 = \mathbb{D}_2 = 2^U$  für eine Menge  $U$  und  $f x = (x \cap a) \cup b$ .  
Offensichtlich ist jedes solche  $f$  monoton
- 2  $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{Z}$  (mit der Ordnung " $\leq$ "). Dann gilt:
  - $\text{inc } x = x + 1$  ist monoton.
  - $\text{dec } x = x - 1$  ist monoton.
  - $\text{inv } x = -x$  ist **nicht monoton**



# Fixpunktiteration

**Gesucht:** möglichst **kleine** Lösung für:

$$x_i \supseteq f_i(x_1, \dots, x_n), \quad i = 1, \dots, n$$

wobei alle  $f_i : \mathbb{D}^n \rightarrow \mathbb{D}$  monoton sind.

# Fixpunktiteration

**Gesucht:** möglichst **kleine** Lösung für:

$$x_i \sqsupseteq f_i(x_1, \dots, x_n), \quad i = 1, \dots, n$$

wobei alle  $f_i : \mathbb{D}^n \rightarrow \mathbb{D}$  monoton sind.

**Idee:**

- Betrachte  $F : \mathbb{D}^n \rightarrow \mathbb{D}^n$  mit  
 $F(x_1, \dots, x_n) = (y_1, \dots, y_n)$  wobei  $y_i = f_i(x_1, \dots, x_n)$ .

# Fixpunktiteration

**Gesucht:** möglichst **kleine** Lösung für:

$$x_i \supseteq f_i(x_1, \dots, x_n), \quad i = 1, \dots, n$$

wobei alle  $f_i : \mathbb{D}^n \rightarrow \mathbb{D}$  monoton sind.

**Idee:**

- Betrachte  $F : \mathbb{D}^n \rightarrow \mathbb{D}^n$  mit  
 $F(x_1, \dots, x_n) = (y_1, \dots, y_n)$  wobei  $y_i = f_i(x_1, \dots, x_n)$ .
- Sind alle  $f_i$  monoton, dann auch  $F$

# Fixpunktiteration

**Gesucht:** möglichst **kleine** Lösung für:

$$x_i \sqsubseteq f_i(x_1, \dots, x_n), \quad i = 1, \dots, n$$

wobei alle  $f_i : \mathbb{D}^n \rightarrow \mathbb{D}$  monoton sind.

**Idee:**

- Betrachte  $F : \mathbb{D}^n \rightarrow \mathbb{D}^n$  mit  $F(x_1, \dots, x_n) = (y_1, \dots, y_n)$  wobei  $y_i = f_i(x_1, \dots, x_n)$ .
- Sind alle  $f_i$  monoton, dann auch  $F$
- Wir **approximieren** sukzessive eine Lösung. Wir konstruieren:

$$\underline{\perp}, \quad F \underline{\perp}, \quad F^2 \underline{\perp}, \quad F^3 \underline{\perp}, \quad \dots$$

**Hoffnung:** Wir erreichen irgendwann eine Lösung ... ?

# Fixpunktiteration

Beispiel:  $\mathbb{D} = 2^{\{a,b,c\}}$ ,  $\sqsubseteq = \subseteq$

$$\begin{aligned}x_1 &\supseteq \{a\} \cup x_3 \\x_2 &\supseteq x_3 \cap \{a, b\} \\x_3 &\supseteq x_1 \cup \{c\}\end{aligned}$$

# Fixpunktiteration

Beispiel:  $\mathbb{D} = 2^{\{a,b,c\}}$ ,  $\sqsubseteq = \subseteq$

$$\begin{aligned}x_1 &\supseteq \{a\} \cup x_3 \\x_2 &\supseteq x_3 \cap \{a, b\} \\x_3 &\supseteq x_1 \cup \{c\}\end{aligned}$$

Die Iteration:

	0	1	2	3	4
$x_1$	$\emptyset$				
$x_2$	$\emptyset$				
$x_3$	$\emptyset$				

# Fixpunktiteration

Beispiel:  $\mathbb{D} = 2^{\{a,b,c\}}$ ,  $\sqsubseteq = \subseteq$

$$\begin{aligned}x_1 &\supseteq \{a\} \cup x_3 \\x_2 &\supseteq x_3 \cap \{a, b\} \\x_3 &\supseteq x_1 \cup \{c\}\end{aligned}$$

Die Iteration:

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

# Fixpunktiteration

Beispiel:  $\mathbb{D} = 2^{\{a,b,c\}}$ ,  $\sqsubseteq = \subseteq$

$$\begin{aligned}x_1 &\supseteq \{a\} \cup x_3 \\x_2 &\supseteq x_3 \cap \{a, b\} \\x_3 &\supseteq x_1 \cup \{c\}\end{aligned}$$

Die Iteration:

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



# Fixpunktiteration

Beispiel:  $\mathbb{D} = 2^{\{a,b,c\}}$ ,  $\sqsubseteq = \subseteq$

$$\begin{aligned}x_1 &\supseteq \{a\} \cup x_3 \\x_2 &\supseteq x_3 \cap \{a, b\} \\x_3 &\supseteq x_1 \cup \{c\}\end{aligned}$$

Die Iteration:

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

# Fixpunktiteration

Beispiel:  $\mathbb{D} = 2^{\{a,b,c\}}$ ,  $\sqsubseteq = \subseteq$

$$\begin{aligned}x_1 &\supseteq \{a\} \cup x_3 \\x_2 &\supseteq x_3 \cap \{a, b\} \\x_3 &\supseteq x_1 \cup \{c\}\end{aligned}$$

Die Iteration:

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

# Fixpunktiteration

Offenbar gilt:

- Gilt  $F^k \underline{\perp} = F^{k+1} \underline{\perp}$ , ist eine Lösung gefunden
- $\underline{\perp}, F \underline{\perp}, F^2 \underline{\perp}, \dots$  bilden eine **aufsteigende Kette** :

$$\underline{\perp} \sqsubseteq F \underline{\perp} \sqsubseteq F^2 \underline{\perp} \sqsubseteq \dots$$

- Sind **alle** aufsteigenden Ketten endlich, gibt es  **$k$  immer**.

# Fixpunktiteration

Offenbar gilt:

- Gilt  $F^k \underline{x} = F^{k+1} \underline{x}$ , ist eine Lösung gefunden
- $\underline{x}, F \underline{x}, F^2 \underline{x}, \dots$  bilden eine **aufsteigende Kette** :

$$\underline{x} \sqsubseteq F \underline{x} \sqsubseteq F^2 \underline{x} \sqsubseteq \dots$$

- Sind **alle** aufsteigenden Ketten endlich, gibt es  **$k$  immer**.

Die zweite Aussage folgt mit **vollständiger Induktion**:

# Fixpunktiteration

Offenbar gilt:

- Gilt  $F^k \underline{x} = F^{k+1} \underline{x}$ , ist eine Lösung gefunden
- $\underline{x}, F \underline{x}, F^2 \underline{x}, \dots$  bilden eine **aufsteigende Kette** :

$$\underline{x} \subseteq F \underline{x} \subseteq F^2 \underline{x} \subseteq \dots$$

- Sind **alle** aufsteigenden Ketten endlich, gibt es  **$k$  immer**.

Die zweite Aussage folgt mit **vollständiger Induktion**:

**Anfang:**  $F^0 \underline{x} = \underline{x} \subseteq F^1 \underline{x}$

# Fixpunktiteration

Offenbar gilt:

- Gilt  $F^k \underline{\perp} = F^{k+1} \underline{\perp}$ , ist eine Lösung gefunden
- $\underline{\perp}, F \underline{\perp}, F^2 \underline{\perp}, \dots$  bilden eine **aufsteigende Kette** :

$$\underline{\perp} \sqsubseteq F \underline{\perp} \sqsubseteq F^2 \underline{\perp} \sqsubseteq \dots$$

- Sind **alle** aufsteigenden Ketten endlich, gibt es  **$k$  immer**.

Die zweite Aussage folgt mit **vollständiger Induktion**:

**Anfang:**  $F^0 \underline{\perp} = \underline{\perp} \sqsubseteq F^1 \underline{\perp}$

**Schluss:** Induktionsannahme:  $F^{i-1} \underline{\perp} \sqsubseteq F^i \underline{\perp}$ . Dann

$$F^i \underline{\perp} = F(F^{i-1} \underline{\perp}) \sqsubseteq F(F^i \underline{\perp}) = F^{i+1} \underline{\perp}$$

da  $F$  monoton ist

# Fixpunktiteration

## Fazit:

Wenn  $\mathbb{D}$  endlich ist, finden wir über Fixpunktiteration mit Sicherheit eine Lösung

## Fragen:

- 1 Gibt es eine kleinste Lösung?

# Fixpunktiteration

## Fazit:

Wenn  $\mathbb{D}$  endlich ist, finden wir über Fixpunktiteration mit Sicherheit eine Lösung

## Fragen:

- 1 Gibt es eine **kleinste** Lösung?
- 2 Wenn ja: findet Iteration die **kleinste** Lösung?



# Fixpunktiteration

## Fazit:

Wenn  $\mathbb{D}$  endlich ist, finden wir über Fixpunktiteration mit Sicherheit eine Lösung

## Fragen:

- 1 Gibt es eine **kleinste** Lösung?
- 2 Wenn ja: findet Iteration die **kleinste** Lösung?
- 3 Was, wenn  $\mathbb{D}$  nicht endlich ist?

# Kleinster Fixpunkt

## Satz: Kleene

In einer **vollständigen** Halbordnung  $\mathbb{D}$  hat jede **stetige** Funktion  $f : \mathbb{D} \rightarrow \mathbb{D}$  einen **kleinsten Fixpunkt**  $d_0$ .

Dieser ist gegeben durch  $d_0 = \bigsqcup_{k \geq 0} f^k \perp$ .

# Kleinster Fixpunkt

## Satz: Kleene

In einer **vollständigen** Halbordnung  $\mathbb{D}$  hat jede **stetige** Funktion  $f : \mathbb{D} \rightarrow \mathbb{D}$  einen **kleinsten Fixpunkt**  $d_0$ .

Dieser ist gegeben durch  $d_0 = \bigsqcup_{k \geq 0} f^k \perp$ .

## Bemerkung:

- Eine Funktion  $f$  heißt **stetig**, falls für jede aufsteigende Kette  $d_0 \sqsubseteq \dots \sqsubseteq d_m \sqsubseteq \dots$  gilt:  $f(\bigsqcup_{m \geq 0} d_m) = \bigsqcup_{m \geq 0} (f d_m)$ .
- Werden alle aufsteigenden Ketten irgendwann **stabil**, ist jede monotone Funktion automatisch stetig

# Kleinster Fixpunkt

## Satz: Kleene

In einer **vollständigen** Halbordnung  $\mathbb{D}$  hat jede **stetige** Funktion  $f : \mathbb{D} \rightarrow \mathbb{D}$  einen **kleinsten Fixpunkt**  $d_0$ .

Dieser ist gegeben durch  $d_0 = \bigsqcup_{k \geq 0} f^k \perp$ .

## Bemerkung:

- Eine Funktion  $f$  heißt **stetig**, falls für jede aufsteigende Kette  $d_0 \sqsubseteq \dots \sqsubseteq d_m \sqsubseteq \dots$  gilt:  $f(\bigsqcup_{m \geq 0} d_m) = \bigsqcup_{m \geq 0} (f d_m)$ .
- Werden alle aufsteigenden Ketten irgendwann **stabil**, ist jede monotone Funktion automatisch stetig
- Eine Halbordnung heißt **vollständig (CPO)**, falls alle aufsteigenden Ketten kleinste obere Schranken haben
- Jeder vollständige Verband ist auch eine vollständige Halbordnung

# Kleinster Fixpunkt

## Satz: Kleene

In einer **vollständigen** Halbordnung  $\mathbb{D}$  hat jede **stetige** Funktion  $f : \mathbb{D} \rightarrow \mathbb{D}$  einen **kleinsten Fixpunkt**  $d_0 = \bigsqcup_{k \geq 0} f^k \perp$ .

## Beweis:

$$\begin{aligned} (1) \quad f d_0 = d_0 : \quad f d_0 &= f \left( \bigsqcup_{m \geq 0} (f^m \perp) \right) \\ &= \bigsqcup_{m \geq 0} (f^{m+1} \perp) \quad \text{wegen Stetigkeit} \\ &= \perp \sqcup \left( \bigsqcup_{m \geq 0} (f^{m+1} \perp) \right) \\ &= \bigsqcup_{m \geq 0} (f^m \perp) \\ &= d_0 \end{aligned}$$

(2)  $d_0$  ist **kleinster** Fixpunkt:

Sei  $f d_1 = d_1$  weiterer Fixpunkt. Wir zeigen:  $\forall m \geq 0 : f^m \perp \sqsubseteq d_1$ .

$$\begin{aligned} m = 0 : \quad & \perp \sqsubseteq d_1 \quad \text{nach Definition} \\ m > 0 : \quad & \text{Gelte } f^{m-1} \perp \sqsubseteq d_1 \quad \text{Dann folgt:} \\ & f^m \perp = f (f^{m-1} \perp) \\ & \sqsubseteq f d_1 \quad \text{wegen Monotonie} \\ & = d_1 \end{aligned}$$

# Kleinsten Fixpunkt

## Bemerkung:

- Jede **stetige** Funktion ist auch monoton
- Betrachte die Menge:  $P = \{x \in \mathbb{D} \mid x \sqsupseteq f x\}$   
Der kleinste Fixpunkt  $d_0$  ist in  $P$  und **untere Schranke**  
 $\implies d_0$  ist der kleinste Wert  $x$  mit  $x \sqsupseteq f x$

# Kleinster Fixpunkt

## Bemerkung:

- Jede **stetige** Funktion ist auch monoton
- Betrachte die Menge:  $P = \{x \in \mathbb{D} \mid x \sqsupseteq f x\}$   
Der kleinste Fixpunkt  $d_0$  ist in  $P$  und **untere Schranke**  
 $\implies d_0$  ist der kleinste Wert  $x$  mit  $x \sqsupseteq f x$

## Anwendung:

Sei  $x_i \sqsupseteq f_i(x_1, \dots, x_n)$ ,  $i = 1, \dots, n$  (\*) ein **Ungleichungssystem**,  
wobei alle  $f_i : \mathbb{D}^n \rightarrow \mathbb{D}$  monoton sind.

# Kleinster Fixpunkt

## Bemerkung:

- Jede **stetige** Funktion ist auch monoton
- Betrachte die Menge:  $P = \{x \in \mathbb{D} \mid x \sqsupseteq f x\}$   
Der kleinste Fixpunkt  $d_0$  ist in  $P$  und **untere Schranke**  
 $\implies d_0$  ist der kleinste Wert  $x$  mit  $x \sqsupseteq f x$

## Anwendung:

Sei  $x_i \sqsupseteq f_i(x_1, \dots, x_n)$ ,  $i = 1, \dots, n$  (\*) ein **Ungleichungssystem**,  
wobei alle  $f_i : \mathbb{D}^n \rightarrow \mathbb{D}$  monoton sind.

$\implies$  kleinste Lösung von (\*)  $\equiv$  kleinster Fixpunkt von  $F$



## Kleinstes Fixpunkt – für $\text{First}_k$

Der Kleenesche Fixpunkt-Satz liefert uns nicht nur die **Existenz** einer kleinsten Lösung sondern auch eine **Charakterisierung**

### Satz:

Die Mengen  $\text{First}_k(\{w \in T^* \mid A \rightarrow^* w\})$ ,  $A \in N$ , sind die kleinste Lösung des Ungleichungssystems:

$$\text{First}_k(A) \supseteq \text{First}_k(X_1) \odot \dots \odot \text{First}_k(X_m), \quad A \rightarrow X_1 \dots X_m \in P$$

## Kleinsten Fixpunkt – für $\text{First}_k$

Der Kleenesche Fixpunkt-Satz liefert uns nicht nur die **Existenz** einer kleinsten Lösung sondern auch eine **Charakterisierung**

### Satz:

Die Mengen  $\text{First}_k(\{w \in T^* \mid A \rightarrow^* w\})$ ,  $A \in N$ , sind die kleinste Lösung des Ungleichungssystems:

$$\text{First}_k(A) \supseteq \text{First}_k(X_1) \odot \dots \odot \text{First}_k(X_m), \quad A \rightarrow X_1 \dots X_m \in P$$

**Beweis-Idee:** Sei  $F^{(m)}(A)$  die  $m$ -te Approximation an den Fixpunkt.

- 1 Falls  $A \rightarrow^m u$ , dann  $\text{First}_k(u) \subseteq F^{(m)}(A)$ .
- 2 Falls  $w \in F^{(m)}(A)$ , dann  $A \rightarrow^* u$  für  $u \in T^*$  mit  $\text{First}_k(u) = \{w\}$

## Fixpunktiteration – für $\text{First}_k$

### Fazit:

Wir können  $\text{First}_k$  durch Fixpunkt-Iteration berechnen, d.h. durch wiederholtes Einsetzen.

## Fixpunktiteration – für $\text{First}_k$

### Fazit:

Wir können  $\text{First}_k$  durch Fixpunkt-Iteration berechnen, d.h. durch wiederholtes Einsetzen.

**Achtung:** Naive Fixpunkt-Iteration ist ziemlich ineffizient

**Idee:** Round Robin Iteration

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

# Round-Robin-Iteration

Unser Mini-Beispiel:  $\mathbb{D} = 2^{\{a,b,c\}}$ ,  $\sqsubseteq = \subseteq$

$$\begin{aligned}x_1 &\supseteq \{a\} \cup x_3 \\x_2 &\supseteq x_3 \cap \{a, b\} \\x_3 &\supseteq x_1 \cup \{c\}\end{aligned}$$

Die Round-Robin-Iteration:

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

## Round-Robin-Iteration

Der Code für Round Robin Iteration sieht so aus:

```
for ( $i = 1; i \leq n; i++$ )  $x_i = \perp$ ;  
do {  
     $finished = true$ ;  
    for ( $i = 1; i \leq n; i++$ ) {  
         $new = f_i(x_1, \dots, x_n)$ ;  
        if ( $!(x_i \sqsupseteq new)$ ) {  
             $finished = false$ ;  
             $x_i = x_i \sqcup new$ ;  
        }  
    }  
} while ( $!finished$ );
```

# Round-Robin-Iteration

## Zur Korrektheit:

Sei  $y_i^{(d)}$  die  $i$ -te Komponente von  $F^d \underline{1}$ .

Sei  $x_i^{(d)}$  der Wert von  $x_i$  nach der  $i$ -ten RR-Iteration.

Man zeigt:

$$\bullet y_i^{(d)} \sqsubseteq x_i^{(d)}$$

# Round-Robin-Iteration

## Zur Korrektheit:

Sei  $y_i^{(d)}$  die  $i$ -te Komponente von  $F^d \underline{1}$ .

Sei  $x_i^{(d)}$  der Wert von  $x_i$  nach der  $i$ -ten RR-Iteration.

Man zeigt:

- 1  $y_i^{(d)} \sqsubseteq x_i^{(d)}$
- 2  $x_i^{(d)} \sqsubseteq z_i$  für jede Lösung  $(z_1, \dots, z_n)$



# Round-Robin-Iteration

## Zur Korrektheit:

Sei  $y_i^{(d)}$  die  $i$ -te Komponente von  $F^d \underline{x}$ .

Sei  $x_i^{(d)}$  der Wert von  $x_i$  nach der  $i$ -ten RR-Iteration.

Man zeigt:

- 1  $y_i^{(d)} \subseteq x_i^{(d)}$
- 2  $x_i^{(d)} \subseteq z_i$  für jede Lösung  $(z_1, \dots, z_n)$
- 3 Terminiert RR-Iteration nach  $d$  Runden, ist  $(x_1^{(d)}, \dots, x_n^{(d)})$  eine Lösung

## Round-Robin-Iteration – für $\text{First}_2$

$$\begin{aligned}\text{First}_2(E) &\supseteq \text{First}_2(E) \odot \{+\} \odot \text{First}_2(T) \cup \text{First}_2(T) \\ \text{First}_2(T) &\supseteq \text{First}_2(T) \odot \{*\} \odot \text{First}_2(F) \cup \text{First}_2(F) \\ \text{First}_2(F) &\supseteq \{(\} \odot \text{First}_2(E) \odot \{)\} \cup \{\text{name, int}\}\end{aligned}$$

### Die RR-Iteration:

$\text{First}_2$	1	2	3
$F$	name, int	( name, ( int	((
$T$	name, int	( name, ( int, name *, int *	((
$E$	name, int	( name, ( int, name *, int *, name +, int +	((

Der Einfachheit halber haben wir in jeder Iteration nur die **neuen** Elemente vermerkt.

## Round-Robin-Iteration – für First<sub>2</sub>

### Diskussion:

- Die Länge  $h$  der längsten echt aufsteigenden Kette nennen wir auch **Höhe** von  $\mathbb{D} \dots$
- Im Falle von First <sub>$k$</sub>  ist die Höhe des Verbands **exponentiell** in  $k$
- Die Anzahl der Runden von RR-Iteration ist beschränkt durch  $\mathcal{O}(n \cdot h)$  ( $n$  die Anzahl der Variablen)
- Die **praktische** Effizienz von RR-Iteration hängt allerdings auch von der **Anordnung** der Variablen ab
- Anstelle von RR-Iteration gibt es auch schnellere Fixpunkt-Verfahren, die aber im schlimmsten Fall immer noch exponentiell sind

$\implies$  Man beschränkt sich i.a. auf **kleine  $k$  !**

# **Kapitel 6:**

## **Top-down Parsing**

# Topdown Parsing

## Idee:

- Benutze den Item-Kellerautomaten.
- Benutze die nächsten  $k$  Zeichen, um die Regeln für die Expansionen zu bestimmen
- Eine Grammatik heißt  $LL(k)$ , falls dies immer eindeutig möglich ist.

# Topdown Parsing

## Idee:

- Benutze den Item-Kellerautomaten.
- Benutze die nächsten  $k$  Zeichen, um die Regeln für die Expansionen zu bestimmen
- Eine Grammatik heißt  $LL(k)$ , falls dies immer eindeutig möglich ist.

## Definition:

Eine reduzierte kontextfreie Grammatik heißt dann  $LL(k)$ , falls für je zwei verschiedene Regeln  $A \rightarrow \alpha$ ,  $A \rightarrow \alpha' \in P$  und jede Linksableitung  $S \rightarrow_L^* u A \beta$  mit  $u \in T^*$  gilt:

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

$LL(k)$  steht für Left-to-right parsing, Leftmost derivation, und Vorrorausschau der Länge  $k$ .

# Topdown Parsing

## Beispiel 1:

$S \rightarrow$  if (  $E$  )  $S$  else  $S$  |  
while (  $E$  )  $S$  |  
 $E$  ;  
 $E \rightarrow$  id

ist  $LL(1)$ , da  $\text{First}_1(E) = \{\text{id}\}$

# Topdown Parsing

## Beispiel 1:

```
 $S \rightarrow \text{if } ( E ) S \text{ else } S \mid$   
 $\text{while } ( E ) S \mid$   
 $E;$   
 $E \rightarrow \text{id}$ 
```

ist  $LL(1)$ , da  $\text{First}_1(E) = \{\text{id}\}$

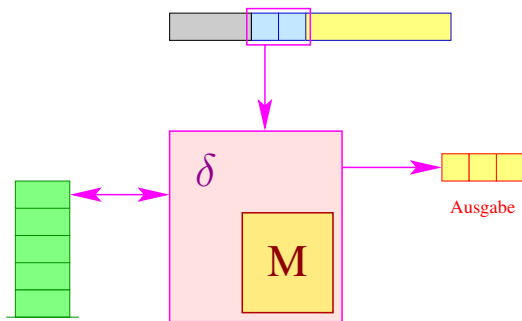
## Beispiel 2:

```
 $S \rightarrow \text{if } ( E ) S \text{ else } S \mid$   
 $\text{if } ( E ) S \mid$   
 $\text{while } ( E ) S \mid$   
 $E;$   
 $E \rightarrow \text{id}$ 
```

... ist nicht  $LL(k)$  für jedes  $k > 0$ .



## Struktur des $LL(k)$ -Parsers:



- Der Parser sieht ein Fenster der Länge  $k$  der Eingabe;
- er realisiert im Wesentlichen den Item-Kellerautomaten;
- die Tabelle  $M[q, w]$  enthält die jeweils zuwählende Regel

# Topdown Parsing

... im Beispiel:

$S \rightarrow \text{if } ( E ) S \text{ else } S^0 \mid$   
 $\text{while } ( E ) S^1 \mid$   
 $E;^2$   
 $E \rightarrow \text{id}^0$

**Zustände:** Items

**Tabelle:**

	if	while	id
$[\dots \rightarrow \dots \bullet S \dots]$	0	1	2
$[\dots \rightarrow \dots \bullet E \dots]$	—	—	0

# Topdown Parsing

## Im Allgemeinen ...

- ist die Menge der möglichen nächsten  $k$  Zeichen gegeben durch:

$$\text{First}_k(\alpha\beta) = \text{First}_k(\alpha) \odot \text{First}_k(\beta)$$

wobei:

- (1)  $\alpha$  die rechte Seite der passenden Regel;
  - (2)  $\beta$  ein möglicher rechter Kontext von  $A$  ist
- $\text{First}_k(\beta)$  müssen wir **dynamisch** akkumulieren.

$\implies$  Wir erweitern Items um Vorausschau-Mengen

...

## Erweiterte Items

Ein **erweitertes Item** ist ein Paar:  $[A \rightarrow \alpha \bullet \gamma, L]$  ( $A \rightarrow \alpha \gamma \in P, L \subseteq T^{\leq k}$ )

Die Menge  $L$  benutzen wir, um  $\text{First}_k(\beta)$  für den rechten Kontext  $\beta$  von  $A$  zu repräsentieren

# Erweiterte Items

Ein **erweitertes Item** ist ein Paar:  $[A \rightarrow \alpha \bullet \gamma, L]$  ( $A \rightarrow \alpha \gamma \in P, L \subseteq T^{\leq k}$ )

Die Menge  $L$  benutzen wir, um  $\text{First}_k(\beta)$  für den rechten Kontext  $\beta$  von  $A$  zu repräsentieren

## Konstruktion:

**Zustände:** erweiterte Items

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

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

**Übergänge:**

# Erweiterte Items

Ein **erweitertes Item** ist ein Paar:  $[A \rightarrow \alpha \bullet \gamma, L]$  ( $A \rightarrow \alpha \gamma \in P, L \subseteq T^{\leq k}$ )

Die Menge  $L$  benutzen wir, um  $\text{First}_k(\beta)$  für den rechten Kontext  $\beta$  von  $A$  zu repräsentieren

## Konstruktion:

**Zustände:** erweiterte Items

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

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

**Übergänge:**

**Expansion:**  $([A \rightarrow \alpha \bullet B \beta, L], \epsilon, [A \rightarrow \alpha \bullet B \beta, L] [B \rightarrow \bullet \gamma, \text{First}_k(\beta) \odot L])$

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

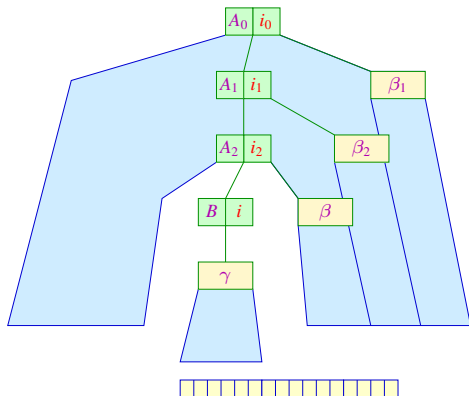
**Shift:**  $([A \rightarrow \alpha \bullet a \beta, L], a, [A \rightarrow \alpha a \bullet \beta, L])$

für  $A \rightarrow \alpha a \beta \in P$

**Reduce:**  $([A \rightarrow \alpha \bullet B \beta, L] [B \rightarrow \gamma \bullet, L'], \epsilon, [A \rightarrow \alpha B \bullet \beta, L])$

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

# Vorausschau-Tabelle



Die Vorausschau-Tabelle: Wir setzen

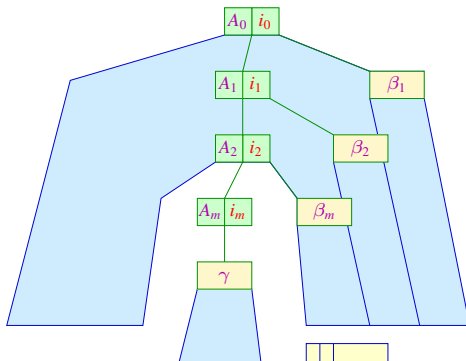
$$M[[A \rightarrow \alpha \bullet B \beta, L], w] = \{i \mid (B, i) = (B \rightarrow \gamma), w \in \text{First}_k(\gamma) \odot \text{First}_k(\beta) \odot L\}$$

# Vorausschau-Tabelle

$$\begin{aligned}
 ([A_0 \rightarrow \bullet \alpha_1 A_1 \beta_1, L_1], uv) &\vdash^* ([A_0 \rightarrow \alpha_1 \bullet A_1 \beta_1, L_1] \dots [A_{m-1} \rightarrow \alpha_m \bullet A_m \beta_m, L_m], v) \\
 &\vdash^* ([A_0 \rightarrow \alpha_1 A_1 \beta_1 \bullet, L_1], \epsilon)
 \end{aligned}$$

... gilt genau dann wenn:

- 1  $\alpha_1 \dots \alpha_m \rightarrow^* u$
- 2  $A_m \beta_m \dots \beta_1 \rightarrow^* v$
- 3  $L_m = \text{First}_k(\beta_{m-1}) \odot \dots \odot \text{First}_k(\beta_1) \odot L_1$





# LL(k)-Grammatik

## Satz:

Die reduzierte kontextfreie Grammatik  $G$  ist  $LL(k)$  genau dann wenn für jedes Eingabewort zu jedem Zeitpunkt in der Berechnung des erweiterten Item-Kellerautomaten  $|M[[A \rightarrow \alpha \bullet B \beta, L], w]| \leq 1$  gilt.

Hierbei ist  $[A \rightarrow \alpha \bullet B \beta, L]$  das aktuelle oberste Kellersymbol und  $w$  aus den nächsten  $k$  Zeichen (oder  $k' < k$  Zeichen, falls der noch zu lesende Teil der Eingabe Länge  $k' < k$  hat) der Eingabe besteht.

## Diskussion:

- Der erweiterte Item-Kellerautomat zusammen mit einer  $k$ -Vorausschau-Tabelle erlaubt die deterministische Rekonstruktion einer Links-Ableitung für eine  $LL(k)$  Grammatik.
- Die Anzahl der Vorausschau-Mengen  $L$  kann sehr groß sein

# LL(k)-Grammatik

Beispiel:  $S \rightarrow \epsilon \mid aSb$

Die Übergänge des erweiterten Item-Kellerautomaten ( $k = 1$ ):

0	$[S' \rightarrow \bullet S, \{\epsilon\}]$	$\epsilon$	$[S' \rightarrow \bullet S, \{\epsilon\}] [S \rightarrow \bullet, \{\epsilon\}]$
1	$[S' \rightarrow \bullet S, \{\epsilon\}]$	$\epsilon$	$[S' \rightarrow \bullet S, \{\epsilon\}] [S \rightarrow \bullet aSb, \{\epsilon\}]$
2	$[S \rightarrow \bullet aSb, \{\epsilon\}]$ $[S \rightarrow \bullet aSb, \{b\}]$	$a$ $a$	$[S \rightarrow a \bullet Sb, \{\epsilon\}]$ $[S \rightarrow a \bullet Sb, \{b\}]$
3	$[S \rightarrow a \bullet Sb, \{\epsilon\}]$ $[S \rightarrow a \bullet Sb, \{b\}]$	$\epsilon$ $\epsilon$	$[S \rightarrow a \bullet Sb, \{\epsilon\}] [S \rightarrow \bullet, \{b\}]$ $[S \rightarrow a \bullet Sb, \{b\}] [S \rightarrow \bullet, \{b\}]$
4	$[S \rightarrow a \bullet Sb, \{\epsilon\}]$ $[S \rightarrow a \bullet Sb, \{b\}]$	$\epsilon$ $\epsilon$	$[S \rightarrow a \bullet Sb, \{\epsilon\}] [S \rightarrow \bullet aSb, \{b\}]$ $[S \rightarrow a \bullet Sb, \{b\}] [S \rightarrow \bullet aSb, \{b\}]$
5	$[S \rightarrow a \bullet Sb, \{\epsilon\}] [S \rightarrow \bullet, \{b\}]$ $[S \rightarrow a \bullet Sb, \{b\}] [S \rightarrow \bullet, \{b\}]$	$\epsilon$ $\epsilon$	$[S \rightarrow aS \bullet b, \{\epsilon\}]$ $[S \rightarrow aS \bullet b, \{b\}]$
	...		...

# LL(k)-Grammatik

	...		...
6	$[S \rightarrow a \bullet Sb, \{\epsilon\}]$ $[S \rightarrow aSb\bullet, \{b\}]$ $[S \rightarrow a \bullet Sb, \{b\}]$ $[S \rightarrow aSb\bullet, \{b\}]$	$\epsilon$	$[S \rightarrow aS\bullet b, \{\epsilon\}]$ $[S \rightarrow aS\bullet b, \{b\}]$
7	$[S \rightarrow aS\bullet b, \{\epsilon\}]$ $[S \rightarrow aS\bullet b, \{b\}]$	$b$ $b$	$[S \rightarrow aSb\bullet, \{\epsilon\}]$ $[S \rightarrow aSb\bullet, \{b\}]$
8	$[S' \rightarrow \bullet S, \{\epsilon\}]$ $[S \rightarrow \bullet, \{\epsilon\}]$	$\epsilon$	$[S' \rightarrow S\bullet, \{\epsilon\}]$
9	$[S' \rightarrow \bullet S, \{\epsilon\}]$ $[S \rightarrow aSb\bullet, \{\epsilon\}]$	$\epsilon$	$[S' \rightarrow S\bullet, \{\epsilon\}]$

Die 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

# Starke LL(k)-Grammatiken

## Beobachtung:

- Im letzten Beispiel hängt die auszuwählende Regel nicht von den Erweiterungen der Items ab!
- Unter dieser Voraussetzung können wir den Item-Kellerautomaten **ohne** Erweiterung benutzen.
- Hängt die auszuwählende Regel **nur** von der aktuellen Vorausschau  $w$  ab, nennen wir  $G$  auch **stark LL(k)**.

# Starke LL(k)-Grammatiken

## Beobachtung:

- Im letzten Beispiel hängt die auszuwählende Regel nicht von den Erweiterungen der Items ab!
- Unter dieser Voraussetzung können wir den Item-Kellerautomaten **ohne** Erweiterung benutzen.
- Hängt die auszuwählende Regel **nur** von der aktuellen Vorausschau  $w$  ab, nennen wir  $G$  auch **stark LL(k)**.

## Definition:

$$\text{Follow}_k(A) = \bigcup \{ \text{First}_k(\beta) \mid S \xrightarrow{*}_L u A \beta \}$$

# Starke LL(k)-Grammatiken

## Beobachtung:

- Im letzten Beispiel hängt die auszuwählende Regel nicht von den Erweiterungen der Items ab!
- Unter dieser Voraussetzung können wir den Item-Kellerautomaten **ohne** Erweiterung benutzen.
- Hängt die auszuwählende Regel **nur** von der aktuellen Vorausschau  $w$  ab, nennen wir  $G$  auch **stark LL(k)**.

## Definition:

$$\text{Follow}_k(A) = \bigcup \{ \text{First}_k(\beta) \mid S \rightarrow_L^* u A \beta \}$$

## Definition:

Die reduzierte kontextfreie Grammatik  $G$  heißt **stark LL(k)**, falls für je zwei verschiedene  $A \rightarrow \alpha, A \rightarrow \alpha' \in P$ :

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

# Starke LL(k)-Grammatiken

Im Beispiel:  $S \rightarrow \epsilon \mid aSb$

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

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

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

**Wir schließen:** Die Grammatik ist in der Tat **stark LL(1)**

# Starke LL(k)-Grammatiken

Ist  $G$  eine starke  $LL(k)$ -Grammatik, können wir die Vorausschau-Tabelle statt mit (erweiterten) Items mit Nichtterminalen indizieren.

Wir setzen

$$M[B, w] = \begin{cases} i & \text{falls } (B, i) = (B \rightarrow \gamma) \text{ und } w \in \text{First}_k(\gamma) \odot \text{Follow}_k(B) \\ - & \text{falls solch eine Regel nicht existiert} \end{cases}$$

Im Beispiel:  $S \rightarrow \epsilon \mid aSb$

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



# Starke LL(k)-Grammatiken

Ist  $G$  eine starke  $LL(k)$ -Grammatik, können wir die Vorausschau-Tabelle statt mit (erweiterten) Items mit Nichtterminalen indizieren.

Wir setzen

$$M[B, w] = \begin{cases} i & \text{falls } (B, i) = (B \rightarrow \gamma) \text{ und } w \in \text{First}_k(\gamma) \odot \text{Follow}_k(B) \\ - & \text{falls solch eine Regel nicht existiert} \end{cases}$$

Im Beispiel:  $s \rightarrow \epsilon \mid a S b$

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

## Satz:

- Jede starke  $LL(k)$ -Grammatik ist auch  $LL(k)$ .
- Jede  $LL(1)$ -Grammatik ist bereits stark  $LL(1)$ .

# Starke LL(k)-Grammatiken

## Satz: Teil 1

- Jede starke  $LL(k)$ -Grammatik ist auch  $LL(k)$ .

## Beweis:

Sei  $G$  stark  $LL(k)$ .

Betrachte eine Ableitung  $S \xrightarrow{*}_L u A \beta$  und Regeln  $A \rightarrow \alpha$ ,  $A \rightarrow \alpha' \in P$ .

Dann haben wir:

$$\begin{aligned} \text{First}_k(\alpha\beta) \cap \text{First}_k(\alpha'\beta) &= \text{First}_k(\alpha) \odot \text{First}_k(\beta) \cap \text{First}_k(\alpha') \odot \text{First}_k(\beta) \\ &\subseteq \text{First}_k(\alpha) \odot \text{Follow}_k(A) \cap \text{First}_k(\alpha') \odot \text{Follow}_k(A) \\ &= \emptyset \end{aligned}$$

$\implies$  Folglich ist  $G$  auch  $LL(k)$

□.

# Starke LL(k)-Grammatiken

## Satz: Teil 2

- Jede  $LL(1)$ -Grammatik ist bereits stark  $LL(1)$ .

## Beweis:

Sei  $G$   $LL(1)$ .

Betrachte zwei verschiedene Regeln  $A \rightarrow \alpha, A \rightarrow \alpha' \in P$ .

**Fall 1:**  $\epsilon \in \text{First}_1(\alpha) \cap \text{First}_1(\alpha')$ .

Dann kann  $G$  nicht  $LL(1)$  sein

# Starke LL(k)-Grammatiken

## Satz: Teil 2

- Jede  $LL(1)$ -Grammatik ist bereits stark  $LL(1)$ .

## Beweis:

Sei  $G$   $LL(1)$ .

Betrachte zwei verschiedene Regeln  $A \rightarrow \alpha$ ,  $A \rightarrow \alpha' \in P$ .

**Fall 1:**  $\epsilon \in \text{First}_1(\alpha) \cap \text{First}_1(\alpha')$ .

Dann kann  $G$  nicht  $LL(1)$  sein

**Fall 2:**  $\epsilon \notin \text{First}_1(\alpha) \cup \text{First}_1(\alpha')$ .

Sei  $S \xrightarrow{*}_L u A \beta$ . Da  $G$   $LL(1)$  ist, gilt:

$$\begin{aligned} & \text{First}_1(\alpha) \odot \text{Follow}_1(A) \cap \text{First}_1(\alpha') \odot \text{Follow}_1(A) \\ &= \text{First}_1(\alpha) \cap \text{First}_1(\alpha') \\ &= \text{First}_1(\alpha) \odot \text{First}_1(\beta) \cap \text{First}_1(\alpha') \odot \text{First}_1(\beta) \\ &= \emptyset \end{aligned}$$

# Starke LL(k)-Grammatiken

## Satz: Teil 2

- Jede  $LL(1)$ -Grammatik ist bereits stark  $LL(1)$

## Beweis:

**Fall 3:**  $\epsilon \in \text{First}_1(\alpha)$  und  $\epsilon \notin \text{First}_1(\alpha')$ . Dann gilt:

$$\begin{aligned} & \text{First}_1(\alpha) \odot \text{Follow}_1(A) \cap \text{First}_1(\alpha') \odot \text{Follow}_1(A) \\ &= \text{First}_1(\alpha) \odot \text{Follow}_1(A) \cap \text{First}_1(\alpha') \\ &= \text{First}_1(\alpha) \odot (\bigcup \{ \text{First}_1(\beta) \mid S \xrightarrow{*}_L uA\beta \}) \cap \text{First}_1(\alpha') \\ &= (\bigcup \{ \text{First}_1(\alpha) \odot \text{First}_1(\beta) \mid S \xrightarrow{*}_L uA\beta \}) \cap \text{First}_1(\alpha') \\ &= \bigcup \{ \text{First}_1(\alpha) \odot \text{First}_1(\beta) \cap \text{First}_1(\alpha') \mid S \xrightarrow{*}_L uA\beta \} \\ &= \bigcup \{ \emptyset \mid S \xrightarrow{*}_L uA\beta \} \\ &= \emptyset \end{aligned}$$

# Starke LL(k)-Grammatiken

## Satz: Teil 2

- Jede  $LL(1)$ -Grammatik ist bereits stark  $LL(1)$

## Beweis:

**Fall 3:**  $\epsilon \in \text{First}_1(\alpha)$  und  $\epsilon \notin \text{First}_1(\alpha')$ . Dann gilt:

$$\begin{aligned} & \text{First}_1(\alpha) \odot \text{Follow}_1(A) \cap \text{First}_1(\alpha') \odot \text{Follow}_1(A) \\ &= \text{First}_1(\alpha) \odot \text{Follow}_1(A) \cap \text{First}_1(\alpha') \\ &= \text{First}_1(\alpha) \odot (\bigcup \{ \text{First}_1(\beta) \mid S \rightarrow_L^* uA\beta \}) \cap \text{First}_1(\alpha') \\ &= (\bigcup \{ \text{First}_1(\alpha) \odot \text{First}_1(\beta) \mid S \rightarrow_L^* uA\beta \}) \cap \text{First}_1(\alpha') \\ &= \bigcup \{ \text{First}_1(\alpha) \odot \text{First}_1(\beta) \cap \text{First}_1(\alpha') \mid S \rightarrow_L^* uA\beta \} \\ &= \bigcup \{ \emptyset \mid S \rightarrow_L^* uA\beta \} \\ &= \emptyset \end{aligned}$$

**Fall 4:**  $\epsilon \notin \text{First}_1(\alpha)$  und  $\epsilon \in \text{First}_1(\alpha')$ : analog

# Starke LL(k)-Grammatiken

Beispiel:

$$\begin{aligned} S &\rightarrow aAaa^0 \mid bAb a^1 \\ A &\rightarrow b^0 \mid \epsilon^1 \end{aligned}$$

Offenbar ist die Grammatik  $LL(2)$  – andererseits gilt:

$$\begin{aligned} & \text{First}_2(b) \odot \text{Follow}_2(A) \cap \text{First}_2(\epsilon) \odot \text{Follow}_2(A) \\ &= \{b\} \odot \{aa, ba\} \cap \{\epsilon\} \odot \{aa, ba\} \\ &= \{ba, bb\} \cap \{aa, ba\} \\ &\neq \emptyset \end{aligned}$$

Folglich ist die Grammatik **nicht stark**  $LL(2)$

# Starke LL(k)-Grammatiken

Beispiel:

$$\begin{aligned} S &\rightarrow aAaa^0 \mid bAb a^1 \\ A &\rightarrow b^0 \mid \epsilon^1 \end{aligned}$$

Offenbar ist die Grammatik  $LL(2)$  – andererseits gilt:

$$\begin{aligned} & \text{First}_2(b) \odot \text{Follow}_2(A) \cap \text{First}_2(\epsilon) \odot \text{Follow}_2(A) \\ &= \{b\} \odot \{aa, ba\} \cap \{\epsilon\} \odot \{aa, ba\} \\ &= \{ba, bb\} \cap \{aa, ba\} \\ &\neq \emptyset \end{aligned}$$

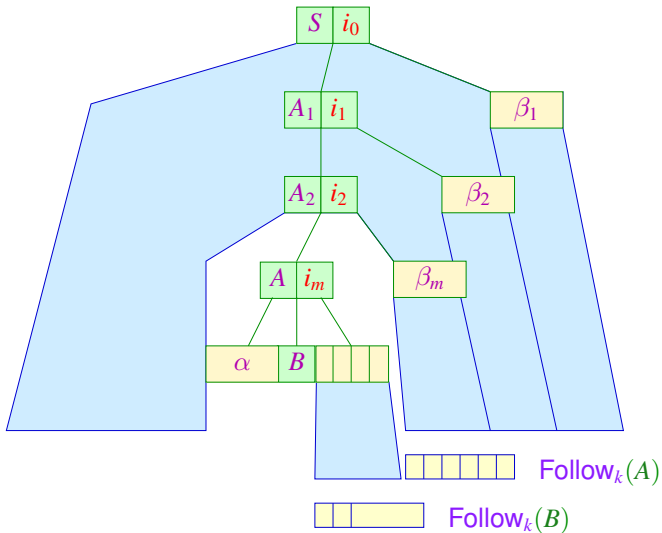
Folglich ist die Grammatik **nicht stark**  $LL(2)$

**Wir schließen:** Für  $k > 1$  ist nicht jede  $LL(k)$ -Grammatik automatisch **stark**  $LL(k)$ .

Ohne Beweis: Zu jeder  $LL(k)$ -Grammatik kann jedoch eine **äquivalente starke**  $LL(k)$ -Grammatik konstruiert werden.



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



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

## Idee:

- 1 Wir stellen ein Ungleichungssystem auf
- 2  $\epsilon$  ist ein möglicher rechter Kontext von  $S$
- 3 Mögliche rechte Kontexte der linken Seite einer Regel propagieren wir ans Ende jeder rechten Seite.

Im Beispiel:  $S \rightarrow \epsilon \mid aSb$

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

$$\text{Follow}_k(S) \supseteq \{b\} \odot \text{Follow}_k(S)$$

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

Allgemein:

$$\begin{aligned}\text{Follow}_k(S) &\supseteq \{\epsilon\} \\ \text{Follow}_k(B) &\supseteq \text{First}_k(X_1) \odot \dots \odot \text{First}_k(X_m) \odot \text{Follow}_k(A) \\ &\text{für } A \rightarrow \alpha B X_1 \dots X_m \in P\end{aligned}$$

Diskussion:

- Man überzeugt sich, dass die **kleinste** Lösung dieses Ungleichungssystems tatsächlich die Mengen  $\text{Follow}_k(B)$  liefert
- Die Größe der auftretenden Mengen steigt mit  $k$  rapide
- In praktischen Systemen wird darum meist nur der Fall  $k = 1$  implementiert ...

# **Kapitel 7:**

## **Schnelle Berechnung von Vorausschau-Mengen**

# Schnelle Berechnung von Vorausschau-Mengen

Im Fall  $k = 1$  lassen sich **First** und **Follow** besonders effizient berechnen

## Beobachtung:

Seien  $L_1, L_2 \subseteq T \cup \{\epsilon\}$  mit  $L_1 \neq \emptyset \neq L_2$ . Dann ist:

$$L_1 \odot L_2 = \begin{cases} L_1 & \text{falls } \epsilon \notin L_1 \\ (L_1 \setminus \{\epsilon\}) \cup L_2 & \text{sonst} \end{cases}$$

Ist  $G$  reduziert, sind alle Mengen  $\text{First}_1(A)$  nichtleer.

# Schnelle Berechnung von Vorausschau-Mengen

## Idee:

- Behandle  $\epsilon$  separat!  
Sei  $\text{empty}(X) = \text{true}$  gdw.  $X \rightarrow^* \epsilon$ .
- Definiere die  $\epsilon$ -freien  $\text{First}_1$ -Mengen

$$\begin{aligned} F_\epsilon(a) &= \{a\} && \text{für } a \in T \\ F_\epsilon(A) &= \text{First}_1(A) \setminus \{\epsilon\} && \text{für } A \in N \end{aligned}$$

## Schnelle Berechnung von Vorausschau-Mengen

Konstruiere direkt ein Ungleichungssystem für  $F_\epsilon(A)$  :

$$F_\epsilon(A) \supseteq F_\epsilon(X_j) \quad \text{falls} \quad A \rightarrow X_1 \dots X_m \in P, \\ \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1})$$

# Schnelle Berechnung von Vorausschau-Mengen

Konstruiere direkt ein Ungleichungssystem für  $F_\epsilon(A)$  :

$$F_\epsilon(A) \supseteq F_\epsilon(X_j) \quad \text{falls} \quad A \rightarrow X_1 \dots X_m \in P, \\ \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1})$$

im Beispiel...

$$\begin{array}{lcl} E & \rightarrow & E + T \quad | \quad T \\ T & \rightarrow & T * F \quad | \quad F \\ F & \rightarrow & ( E ) \quad | \quad \text{name} \quad | \quad \text{int} \end{array}$$

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



# Schnelle Berechnung von Vorausschau-Mengen

Konstruiere direkt ein Ungleichungssystem für  $F_\epsilon(A)$  :

$$F_\epsilon(A) \supseteq F_\epsilon(X_j) \quad \text{falls} \quad A \rightarrow X_1 \dots X_m \in P, \\ \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1})$$

im Beispiel...

$$\begin{array}{lcl} E & \rightarrow & E + T \quad | \quad T \\ T & \rightarrow & T * F \quad | \quad F \\ F & \rightarrow & ( E ) \quad | \quad \text{name} \quad | \quad \text{int} \end{array}$$

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

... erhalten wir:

$$\begin{array}{lcl} F_\epsilon(S') & \supseteq & F_\epsilon(E) \quad F_\epsilon(E) \supseteq F_\epsilon(E) \\ F_\epsilon(E) & \supseteq & F_\epsilon(T) \quad F_\epsilon(T) \supseteq F_\epsilon(T) \\ F_\epsilon(T) & \supseteq & F_\epsilon(F) \quad F_\epsilon(F) \supseteq \{ (, \text{name}, \text{int} ) \} \end{array}$$

# Schnelle Berechnung von Vorausschau-Mengen

Analog dazu das Ungleichungssystem zu  $\text{Follow}_1(A)$  :

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

$$\text{Follow}_1(B) \supseteq F_\epsilon(X_j) \quad \text{falls} \quad A \rightarrow \alpha B X_1 \dots X_m \in P, \\ \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1})$$

$$\text{Follow}_1(B) \supseteq \text{Follow}_1(A) \quad \text{falls} \quad A \rightarrow \alpha B X_1 \dots X_m \in P, \\ \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_m)$$

# Schnelle Berechnung von Vorausschau-Mengen

Analog dazu das Ungleichungssystem zu  $\text{Follow}_1(A)$  :

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

$$\text{Follow}_1(B) \supseteq F_\epsilon(X_j) \quad \text{falls} \quad A \rightarrow \alpha B X_1 \dots X_m \in P, \\ \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1})$$

$$\text{Follow}_1(B) \supseteq \text{Follow}_1(A) \quad \text{falls} \quad A \rightarrow \alpha B X_1 \dots X_m \in P, \\ \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_m)$$

im Beispiel...

$$\begin{array}{lcl} E & \rightarrow & E + T \quad | \quad T \\ T & \rightarrow & T * F \quad | \quad F \\ F & \rightarrow & ( E ) \quad | \quad \text{name} \quad | \quad \text{int} \end{array}$$

# Schnelle Berechnung von Vorausschau-Mengen

Analog dazu das Ungleichungssystem zu  $\text{Follow}_1(A)$  :

$$\begin{aligned} \text{Follow}_1(S) &\supseteq \{\epsilon\} \\ \text{Follow}_1(B) &\supseteq F_\epsilon(X_j) \quad \text{falls} \quad A \rightarrow \alpha B X_1 \dots X_m \in P, \\ &\quad \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1}) \\ \text{Follow}_1(B) &\supseteq \text{Follow}_1(A) \quad \text{falls} \quad A \rightarrow \alpha B X_1 \dots X_m \in P, \\ &\quad \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_m) \end{aligned}$$

im Beispiel...

$$\begin{array}{lcl} E & \rightarrow & E + T \quad | \quad T \\ T & \rightarrow & T * F \quad | \quad F \\ F & \rightarrow & ( E ) \quad | \quad \text{name} \quad | \quad \text{int} \end{array}$$

... erhalten wir:

$$\begin{array}{lcl} \text{Follow}_1(S') &\supseteq &\{\epsilon\} & \text{Follow}_1(E) &\supseteq &\text{Follow}_1(S') \\ \text{Follow}_1(E) &\supseteq &\{+, )\} & \text{Follow}_1(T) &\supseteq &\{*\} \\ \text{Follow}_1(T) &\supseteq &\text{Follow}_1(E) & \text{Follow}_1(F) &\supseteq &\text{Follow}_1(T) \end{array}$$

# Schnelle Berechnung von Vorausschau-Mengen

## Beobachtung:

- Die Form der Ungleichungen dieser Ungleichungssysteme ist:

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

für Variablen  $x, y$  und  $d \in \mathbb{D}$ .

- Solche Ungleichungssysteme heißen **reine Vereinigungs-Probleme**
- Diese Probleme können mit **linearem Aufwand** gelöst werden

im Beispiel:

$$\mathbb{D} = 2^{\{a,b,c\}}$$

$$x_0 \supseteq \{a\}$$

$$x_1 \supseteq \{b\}$$

$$x_2 \supseteq \{c\}$$

$$x_3 \supseteq \{c\}$$

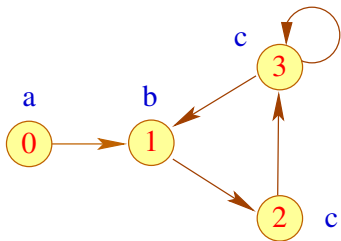
$$x_1 \supseteq x_0$$

$$x_2 \supseteq x_1$$

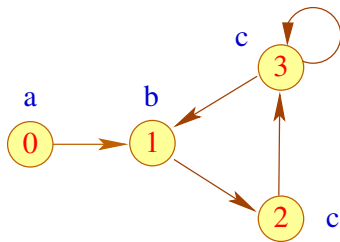
$$x_3 \supseteq x_2$$

$$x_1 \supseteq x_3$$

$$x_3 \supseteq x_3$$



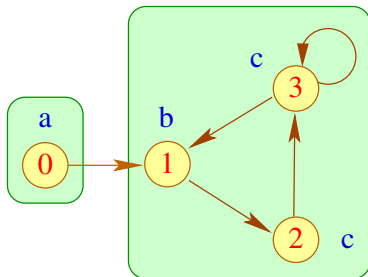
# Schnelle Berechnung von Vorausschau-Mengen



## Vorgehen:

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

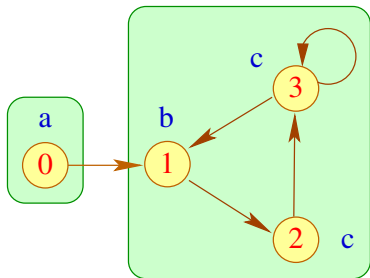
# Schnelle Berechnung von Vorausschau-Mengen



## Vorgehen:

- Konstruiere den **Variablen-Abhängigkeitsgraph** zum Ungleichungssystem.
- Innerhalb einer **starken Zusammenhangskomponente** haben alle Variablen den gleichen Wert

# Schnelle Berechnung von Vorausschau-Mengen

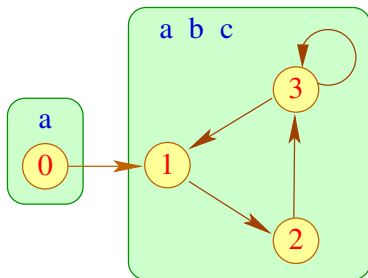


## Vorgehen:

- Konstruiere den **Variablen-Abhängigkeitsgraph** zum Ungleichungssystem.
- Innerhalb einer **starken Zusammenhangskomponente** haben alle Variablen den gleichen Wert
- Hat eine SZK keine eingehenden Kanten, erhält man ihren Wert, indem man die kleinste obere Schranke aller Werte in der SZK berechnet



# Schnelle Berechnung von Vorausschau-Mengen



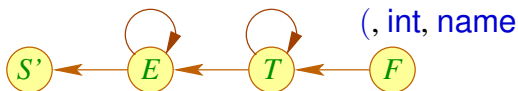
## Vorgehen:

- Konstruiere den **Variablen-Abhängigkeitsgraph** zum Ungleichungssystem.
- Innerhalb einer **starken Zusammenhangskomponente** haben alle Variablen den gleichen Wert
- Hat eine SZK keine eingehenden Kanten, erhält man ihren Wert, indem man die kleinste obere Schranke aller Werte in der SZK berechnet
- Gibt es eingehende Kanten, muss man zusätzlich die Werte an deren Startknoten hinzufügen

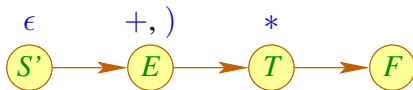
# Schnelle Berechnung von Vorausschau-Mengen

... für unsere Beispiel-Grammatik:

First<sub>1</sub> :



Follow<sub>1</sub> :



# **Kapitel 8:**

## **Bottom-up Analyse**

# Bottom-up Analyse

## Achtung:

Viele Grammatiken sind nicht  $LL(k)$  !

Eine Grund dafür ist:

## Definition

Die Grammatik  $G$  heißt **links-rekursiv**, falls

$$A \rightarrow^+ A\beta \quad \text{für ein } A \in N, \beta \in (T \cup N)^*$$

# Bottom-up Analyse

## Achtung:

Viele Grammatiken sind nicht  $LL(k)$  !

Eine Grund dafür ist:

## Definition

Die Grammatik  $G$  heißt **links-rekursiv**, falls

$$A \rightarrow^+ A\beta \quad \text{für ein } A \in N, \beta \in (T \cup N)^*$$

Beispiel:

$$\begin{array}{lcl} E & \rightarrow & E + T \quad | \quad T \\ T & \rightarrow & T * F \quad | \quad F \\ F & \rightarrow & ( E ) \quad | \quad \text{name} \quad | \quad \text{int} \end{array}$$

... ist links-rekursiv

## Bottom-up Analyse

### Satz:

Ist die Grammatik  $G$  reduziert und links-rekursiv, dann ist  $G$  nicht  $LL(k)$  für jedes  $k$ .

# Bottom-up Analyse

## Satz:

Ist die Grammatik  $G$  reduziert und links-rekursiv, dann ist  $G$  nicht  $LL(k)$  für jedes  $k$ .

**Beweis:** Vereinfachung:  $A \rightarrow A\beta \in P$

$A$  erreichbar  $\implies S \xrightarrow{*}_L uA\gamma \xrightarrow{*}_L uA\beta^n\gamma$  für jedes  $n \geq 0$ .  
 $A$  produktiv  $\implies \exists A \rightarrow \alpha : \alpha \neq A\beta$ .

# Bottom-up Analyse

## Satz:

Ist die Grammatik  $G$  reduziert und links-rekursiv, dann ist  $G$  nicht  $LL(k)$  für jedes  $k$ .

**Beweis:** Vereinfachung:  $A \rightarrow A\beta \in P$

$A$  erreichbar  $\implies S \xrightarrow{*}_L uA\gamma \xrightarrow{*}_L uA\beta^n\gamma$  für jedes  $n \geq 0$ .

$A$  produktiv  $\implies \exists A \rightarrow \alpha : \alpha \neq A\beta$ .

**Annahme:**  $G$  ist  $LL(k)$  Dann gilt für alle  $n \geq 0$ :

$$\text{First}_k(\alpha\beta^n\gamma) \cap \text{First}_k(A\beta\beta^n\gamma) = \emptyset$$

$$\text{Weil } \text{First}_k(\alpha\beta^{n+1}\gamma) \subseteq \text{First}_k(A\beta^{n+1}\gamma)$$

$$\text{folgt: } \text{First}_k(\alpha\beta^n\gamma) \cap \text{First}_k(\alpha\beta^{n+1}\gamma) = \emptyset$$



# Bottom-up Analyse

## Satz:

Ist die Grammatik  $G$  reduziert und links-rekursiv, dann ist  $G$  nicht  $LL(k)$  für jedes  $k$ .

**Beweis:** Vereinfachung:  $A \rightarrow A\beta \in P$

$A$  erreichbar  $\implies S \xrightarrow{*}_L uA\gamma \xrightarrow{*}_L uA\beta^n\gamma$  für jedes  $n \geq 0$ .

$A$  produktiv  $\implies \exists A \rightarrow \alpha : \alpha \neq A\beta$ .

**Annahme:**  $G$  ist  $LL(k)$  Dann gilt für alle  $n \geq 0$ :

$$\text{First}_k(\alpha\beta^n\gamma) \cap \text{First}_k(A\beta\beta^n\gamma) = \emptyset$$

$$\text{Weil } \text{First}_k(\alpha\beta^{n+1}\gamma) \subseteq \text{First}_k(A\beta^{n+1}\gamma)$$

$$\text{folgt: } \text{First}_k(\alpha\beta^n\gamma) \cap \text{First}_k(\alpha\beta^{n+1}\gamma) = \emptyset$$

**Fall 1:**  $\beta \xrightarrow{*} \epsilon$  — Widerspruch !!!

**Fall 2:**  $\beta \xrightarrow{*} w \neq \epsilon \implies$

$$\text{First}_k(\alpha\beta^k\gamma) \cap \text{First}_k(\alpha\beta^{k+1}\gamma) \neq \emptyset$$

# Bottom-up Analyse

**Idee:** Wir rekonstruieren reverse Rechtsableitungen!  
Dazu versuchen wir, für den Shift-Reduce-Parser  $M_G^{(1)}$  die Reduktionsstellen zu identifizieren ...

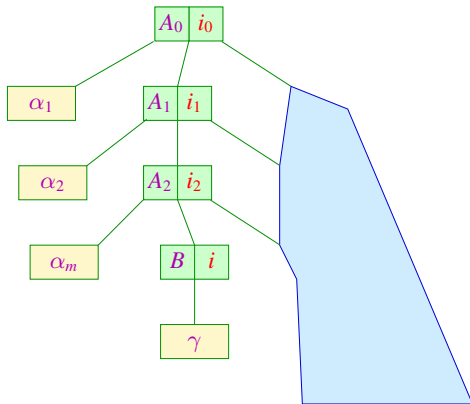
Betrachte eine Berechnung dieses Kellerautomaten:

$$(q_0 \alpha \gamma, v) \vdash (q_0 \alpha B, v) \vdash^* (q_0 S, \epsilon)$$

$\alpha \gamma$  nennen wir **zuverlässiges Präfix** für das vollständige Item  $[B \rightarrow \gamma \bullet]$ .

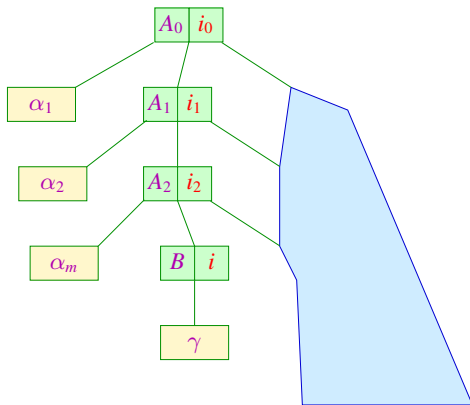
Dann ist  $\alpha \gamma$  zuverlässig für  $[B \rightarrow \gamma \bullet]$  gdw.  $S \xrightarrow{*}_R \alpha B v$

# Bottom-up Analyse



... wobei  $\alpha = \alpha_1 \dots \alpha_m$

# Bottom-up Analyse

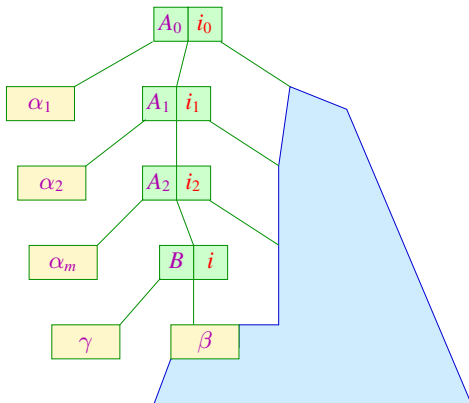


... wobei  $\alpha = \alpha_1 \dots \alpha_m$

Umgekehrt können wir zu jedem möglichen Wort  $\alpha'$  die Menge aller möglicherweise später passenden Regeln ermitteln ...

# Bottom-up Analyse

Das Item  $[B \rightarrow \gamma \bullet \beta]$  heißt **gültig** für  $\alpha'$  gdw.  $S \xrightarrow{*}_R \alpha B v$  mit  $\alpha' = \alpha \gamma$ :



... wobei  $\alpha = \alpha_1 \dots \alpha_m$  :-)

# Charakteristischer Automat

## Beobachtung:

Die Menge der zuverlässigen Präfixe aus  $(N \cup T)^*$  für (vollständige) Items kann mithilfe eines endlichen Automaten berechnet werden:

**Zustände:** Items

**Anfangszustand:**  $[S' \rightarrow \bullet S]$

**Endzustände:**  $\{[B \rightarrow \gamma \bullet] \mid B \rightarrow \gamma \in P\}$

**Übergänge:**

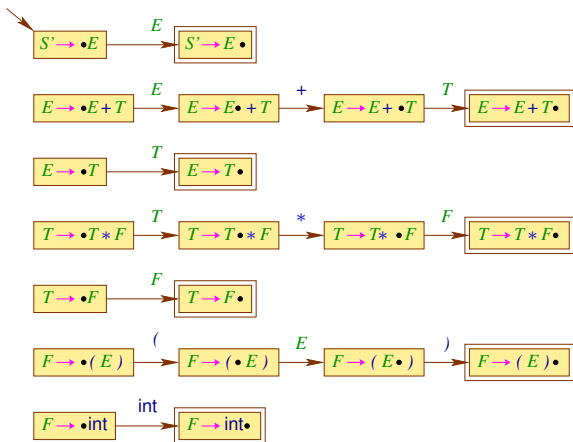
- (1)  $([A \rightarrow \alpha \bullet X \beta], X, [A \rightarrow \alpha X \bullet \beta]), \quad X \in (N \cup T), A \rightarrow \alpha X \beta \in P;$
- (2)  $([A \rightarrow \alpha \bullet B \beta], \epsilon, [B \rightarrow \bullet \gamma]), \quad A \rightarrow \alpha B \beta, B \rightarrow \gamma \in P;$

Den Automaten  $c(G)$  nennen wir **charakteristischen Automaten** für  $G$ .

# Charakteristischer Automat

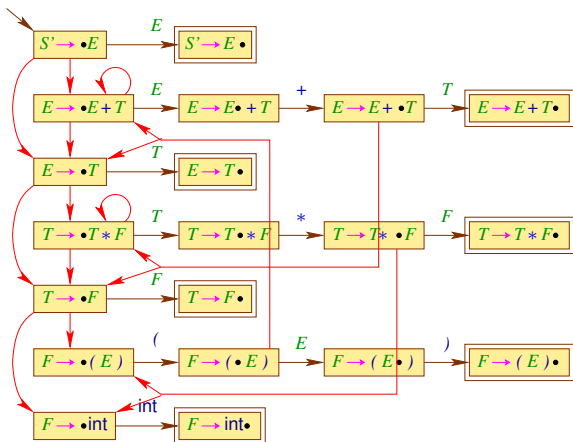
im Beispiel:

$E \rightarrow$	$E + T$		$T$
$T \rightarrow$	$T * F$		$F$
$F \rightarrow$	$( E )$		$\text{int}$



# Charakteristischer Automat

im Beispiel:

$$\begin{array}{lcl} E & \rightarrow & E + T \quad | \quad T \\ T & \rightarrow & T * F \quad | \quad F \\ F & \rightarrow & ( E ) \quad | \quad \text{int} \end{array}$$


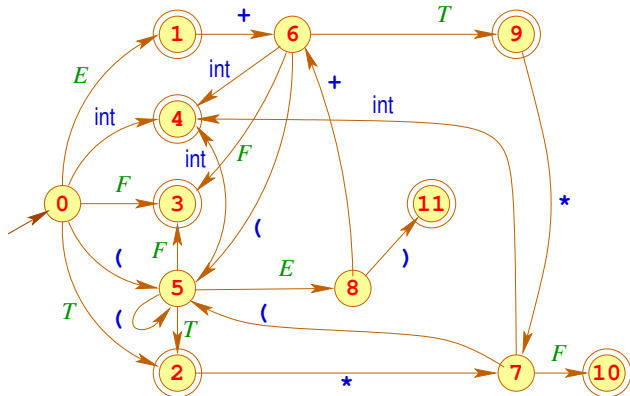


# Kanonischer LR(0)-Automat

Den **kanonischen LR(0)**-Automaten  $LR(G)$  erhalten wir aus  $c(G)$ , indem wir:

- 1 nach jedem lesenden Übergang beliebig viele  $\epsilon$ -Übergänge einschieben (unsere Konstruktion 1 zur Beseitigung von  $\epsilon$ -Übergängen)
- 2 die Teilmengenkonstruktion anwenden.

... im Beispiel:



# Kanonischer LR(0)-Automat

Dazu konstruieren wir:

$$q_0 = \{ [S' \rightarrow \bullet E], \\ [E \rightarrow \bullet E + T], \\ [E \rightarrow \bullet T], \\ [T \rightarrow \bullet T * F], \\ [T \rightarrow \bullet F], \\ [F \rightarrow \bullet (E)], \\ [F \rightarrow \bullet \text{int}] \}$$

$$q_1 = \delta(q_0, E) = \{ [S' \rightarrow E \bullet], \\ [E \rightarrow E \bullet + T] \}$$

$$q_2 = \delta(q_0, T) = \{ [E \rightarrow T \bullet], \\ [T \rightarrow T \bullet * F] \}$$

$$q_3 = \delta(q_0, F) = \{ [T \rightarrow F \bullet] \}$$

$$q_4 = \delta(q_0, \text{int}) = \{ [F \rightarrow \text{int} \bullet] \}$$

# Kanonischer LR(0)-Automat

$$q_5 = \delta(q_0, () = \left\{ \begin{array}{l} [F \rightarrow (\bullet E)], \\ [E \rightarrow \bullet E + T], \\ [E \rightarrow \bullet T], \\ [T \rightarrow \bullet T * F], \\ [T \rightarrow \bullet F], \\ [F \rightarrow \bullet (E)], \\ [F \rightarrow \bullet \text{int}] \end{array} \right\}$$

$$q_6 = \delta(q_1, +) = \left\{ \begin{array}{l} [E \rightarrow E + \bullet T], \\ [T \rightarrow \bullet T * F], \\ [T \rightarrow \bullet F], \\ [F \rightarrow \bullet (E)], \\ [F \rightarrow \bullet \text{int}] \end{array} \right\}$$

$$q_7 = \delta(q_2, *) = \left\{ \begin{array}{l} [T \rightarrow T * \bullet F], \\ [F \rightarrow \bullet (E)], \\ [F \rightarrow \bullet \text{int}] \end{array} \right\}$$

$$q_8 = \delta(q_5, E) = \left\{ \begin{array}{l} [F \rightarrow (E \bullet)] \\ [E \rightarrow E \bullet + T] \end{array} \right\}$$

$$q_9 = \delta(q_6, T) = \left\{ \begin{array}{l} [E \rightarrow E + T \bullet], \\ [T \rightarrow T \bullet * F] \end{array} \right\}$$

$$q_{10} = \delta(q_7, F) = \left\{ [T \rightarrow T * F \bullet] \right\}$$

$$q_{11} = \delta(q_8, )) = \left\{ [F \rightarrow (E) \bullet] \right\}$$

# Kanonischer LR(0)-Automat

## Beachte:

Der kanonische LR(0)-Automat kann auch **direkt** aus der Grammatik konstruiert werden.

Man benötigt die Hilfsfunktion:  $\delta_\epsilon^*$

$$\delta_\epsilon^*(q) = q \cup \{ [B \rightarrow \bullet \gamma] \mid \exists [A \rightarrow \alpha \bullet B' \beta'] \in q, \\ \beta \in (N \cup T)^* : B' \xrightarrow{*} B \beta \}$$

Dann definiert man:

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

# LR(0)-Parser

## Idee zu einem Parser:

- Der Parser verwaltet ein zuverlässiges Präfix  $\alpha = X_1 \dots X_m$  auf dem Keller und benutzt  $LR(G)$ , um Reduktionsstellen zu entdecken.
- Er kann mit einer Regel  $A \rightarrow \gamma$  reduzieren, falls  $[A \rightarrow \gamma \bullet]$  für  $\alpha$  gültig ist
- Damit der Automat nicht immer wieder neu über den Kellerinhalt laufen muss, kellern wir anstelle der  $X_i$  jeweils die **Zustände** !

### Achtung:

Dieser Parser ist nur dann **deterministisch**, wenn jeder Endzustand des kanonischen  $LR(0)$ -Automaten keine **Konflikte** enthält.

# LR(0)-Parser

... im Beispiel:

$$q_1 = \left\{ \begin{array}{l} [S' \rightarrow E \bullet], \\ [E \rightarrow E \bullet + T] \end{array} \right\}$$

$$q_2 = \left\{ \begin{array}{l} [E \rightarrow T \bullet], \\ [T \rightarrow T \bullet * F] \end{array} \right\}$$

$$q_3 = \{ [T \rightarrow F \bullet] \}$$

$$q_4 = \{ [F \rightarrow \text{int} \bullet] \}$$

$$q_9 = \left\{ \begin{array}{l} [E \rightarrow E + T \bullet], \\ [T \rightarrow T \bullet * F] \end{array} \right\}$$

$$q_{10} = \{ [T \rightarrow T * F \bullet] \}$$

$$q_{11} = \{ [F \rightarrow (E) \bullet] \}$$

Die Endzustände  $q_1, q_2, q_9$  enthalten mehr als ein Item

⇒ nicht deterministisch!

# LR(0)-Parser

... im Beispiel:

$$q_1 = \left\{ \begin{array}{l} [S' \rightarrow E \bullet], \\ [E \rightarrow E \bullet + T] \end{array} \right\}$$

$$q_2 = \left\{ \begin{array}{l} [E \rightarrow T \bullet], \\ [T \rightarrow T \bullet * F] \end{array} \right\}$$

$$q_3 = \{ [T \rightarrow F \bullet] \}$$

$$q_4 = \{ [F \rightarrow \text{int} \bullet] \}$$

$$q_9 = \left\{ \begin{array}{l} [E \rightarrow E + T \bullet], \\ [T \rightarrow T \bullet * F] \end{array} \right\}$$

$$q_{10} = \{ [T \rightarrow T * F \bullet] \}$$

$$q_{11} = \{ [F \rightarrow (E) \bullet] \}$$

Die Endzustände  $q_1, q_2, q_9$  enthalten mehr als ein Item

⇒ nicht deterministisch!

Aber wir haben ja auch noch nicht **Vorausschau** eingesetzt

# LR(0)-Parser

## Die Konstruktion des LR(0)-Parsers:

**Zustände:**  $Q \cup \{f\}$  ( $f$  neu)

**Anfangszustand:**  $q_0$

**Endzustand:**  $f$

**Übergänge:**

<b>Shift:</b>	$(p, a, pq)$	falls	$q = \delta(p, a) \neq \emptyset$
<b>Reduce:</b>	$(p q_1 \dots q_m, \epsilon, pq)$	falls	$[A \rightarrow X_1 \dots X_m \bullet] \in q_m,$ $q = \delta(p, A)$
<b>Finish:</b>	$(q_0 p, \epsilon, f)$	falls	$[S' \rightarrow S \bullet] \in p$

wobei  $LR(G) = (Q, T, \delta, q_0, F)$ .



# LR(0)-Parser

## Zur Korrektheit:

### Man zeigt:

Die akzeptierenden Berechnungen des  $LR(0)$ -Parsers stehen in eins-zu-eins Beziehung zu denen des Shift-Reduce-Parsers  $M_G^{(1)}$ .

### Wir folgern:

- ⇒ Die akzeptierte Sprache ist genau  $\mathcal{L}(G)$
- ⇒ Die Folge der Reduktionen einer akzeptierenden Berechnung für ein Wort  $w \in T$  liefert eine **reverse Rechts-Ableitung** von  $G$  für  $w$

# LR(0)-Parser

## Achtung:

Leider ist der LR(0)-Parser im allgemeinen nicht-deterministisch

Wir identifizieren zwei Gründe:

### Reduce-Reduce-Konflikt:

$$[A \rightarrow \gamma \bullet], [A' \rightarrow \gamma' \bullet] \in q \text{ mit } A \neq A' \vee \gamma \neq \gamma'$$

### Shift-Reduce-Konflikt:

$$[A \rightarrow \gamma \bullet], [A' \rightarrow \alpha \bullet a \beta] \in q \text{ mit } a \in T$$

für einen Zustand  $q \in Q$ .

Solche Zustände nennen wir **ungeeignet**.

# LR(0)-Parser

In der Tat gilt:

## Satz:

Die reduzierte Grammatik  $G$  ist genau dann  $LR(0)$  wenn der kanonische  $LR(0)$ -Automat  $LR(G)$  keine ungeeigneten Zustände enthält.

# LR(0)-Parser

In der Tat gilt:

## Satz:

Die reduzierte Grammatik  $G$  ist genau dann  $LR(0)$  wenn der kanonische  $LR(0)$ -Automat  $LR(G)$  keine ungeeigneten Zustände enthält.

## Beweis:

Enthalte  $LR(G)$  einen ungeeigneten Zustand  $q$ .

**Fall 1:**  $[A \rightarrow \gamma \bullet], [A' \rightarrow \gamma' \bullet] \in q$  mit  $A \rightarrow \gamma \neq A' \rightarrow \gamma'$   
Dann gibt es ein zuverlässiges Präfix  $\alpha \gamma = \alpha' \gamma'$  mit

$$S \xrightarrow{*}_R \alpha A w \rightarrow \alpha \gamma w \quad \wedge \quad S \xrightarrow{*}_R \alpha' A' x \rightarrow \alpha' \gamma' x$$

$\implies G$  ist nicht  $LR(0)$

**Fall 2:**  $[A \rightarrow \gamma \bullet], [A' \rightarrow \beta \bullet a \beta'] \in q$

Dann gibt es ein zuverlässiges Präfix  $\alpha \gamma = \alpha' \beta$  mit

$$S \xrightarrow*_R \alpha A w \rightarrow \alpha \gamma w \quad \wedge \quad S \xrightarrow*_R \alpha' A' x \rightarrow \alpha' \beta a \beta' x$$

Ist  $\beta' \in T^*$ , dann ist  $G$  nicht  $LR(0)$

Andernfalls  $\beta' \xrightarrow*_R v_1 X v_2 \rightarrow v_1 u v_2$ . Damit erhalten wir:

$$S \xrightarrow*_R \alpha' \beta a v_1 X v_2 x \rightarrow \alpha' \beta a v_1 u v_2 x$$

$$\implies G \text{ ist nicht } LR(0)$$

**Fall 2:**  $[A \rightarrow \gamma \bullet], [A' \rightarrow \beta \bullet a \beta'] \in q$

Dann gibt es ein zuverlässiges Präfix  $\alpha \gamma = \alpha' \beta$  mit

$$S \xrightarrow*_R \alpha A w \rightarrow \alpha \gamma w \quad \wedge \quad S \xrightarrow*_R \alpha' A' x \rightarrow \alpha' \beta a \beta' x$$

Ist  $\beta' \in T^*$ , dann ist  $G$  nicht  $LR(0)$

Andernfalls  $\beta' \xrightarrow*_R v_1 X v_2 \rightarrow v_1 u v_2$ . Damit erhalten wir:

$$S \xrightarrow*_R \alpha' \beta a v_1 X v_2 x \rightarrow \alpha' \beta a v_1 u v_2 x$$

$$\implies G \text{ ist nicht } LR(0)$$

Enthalte  $LR(G)$  keine ungeeigneten Zustände. Betrachte:

$$S \xrightarrow*_R \alpha A w \rightarrow \alpha \gamma w \quad S \xrightarrow*_R \alpha' A' w' \rightarrow \alpha' \gamma' x$$

Sei  $\delta(q_0, \alpha \gamma) = q$ . Insbesondere ist  $[A \rightarrow \gamma \bullet] \in q$ .

**Annahme:**  $(\alpha, A, w') \neq (\alpha', A', x)$ .

**Fall 1:**  $w' = x$ . Dann muss  $q$   $[A' \rightarrow \gamma' \bullet]$  enthalten

**Fall 2:**  $w' \neq x$ . Weitere Fallunterscheidung

# LR(k)-Grammatik

**Idee:** Benutze  $k$ -Vorausschau, um Konflikte zu lösen.

## Definition:

Die reduzierte kontextfreie Grammatik  $G$  heißt  $LR(k)$ -Grammatik, falls für  $\text{First}_k(w) = \text{First}_k(x)$  aus:

$$\left. \begin{array}{l} S \xrightarrow{*}_R \alpha A w \rightarrow \alpha \beta w \\ S \xrightarrow{*}_R \alpha' A' w' \rightarrow \alpha \beta x \end{array} \right\} \text{folgt: } \alpha = \alpha' \wedge A = A' \wedge w' = x$$

# LR(k)-Grammatik

im Beispiel:

$$(1) \quad S \rightarrow A \mid B \quad A \rightarrow aAb \mid 0 \quad B \rightarrow aBbb \mid 1$$

... ist nicht  $LL(k)$  für jedes  $k$  — aber  $LR(0)$  :

Sei  $S \xrightarrow{*}_R \alpha X w \rightarrow \alpha \beta w$ . Dann ist  $\alpha \underline{\beta}$  von einer der Formen:

$$\underline{A}, \underline{B}, a^n \underline{aAb}, a^n \underline{aBbb}, a^n \underline{0}, a^n \underline{1} \quad (n \geq 0)$$



# LR(k)-Grammatik

im Beispiel:

$$(1) \quad S \rightarrow A \mid B \quad A \rightarrow aAb \mid 0 \quad B \rightarrow aBbb \mid 1$$

... ist nicht  $LL(k)$  für jedes  $k$  — aber  $LR(0)$  :

Sei  $S \xrightarrow{*}_R \alpha X w \rightarrow \alpha \beta w$ . Dann ist  $\alpha \underline{\beta}$  von einer der Formen:

$$\underline{A}, \underline{B}, a^n \underline{aAb}, a^n \underline{aBbb}, a^n \underline{0}, a^n \underline{1} \quad (n \geq 0)$$

$$(2) \quad S \rightarrow aAc \quad A \rightarrow Abb \mid b$$

... ist ebenfalls  $LR(0)$  :

Sei  $S \xrightarrow{*}_R \alpha X w \rightarrow \alpha \beta w$ . Dann ist  $\alpha \underline{\beta}$  von einer der Formen:

$$\underline{ab}, a \underline{Abb}, a \underline{Ac}$$

# LR(k)-Grammatik

im Beispiel:

(3)  $S \rightarrow aAc$      $A \rightarrow bbA \mid b$     ... ist nicht  $LR(0)$ , aber  $LR(1)$ :

Für  $S \xrightarrow{*}_R \alpha X w \rightarrow \alpha \beta w$  mit  $\{y\} = \text{First}_k(w)$  ist  $\alpha \underline{\beta} y$   
von einer der Formen:

$$ab^{2n} \underline{bc}, ab^{2n} \underline{bbAc}, \underline{aAc}$$

# LR(k)-Grammatik

im Beispiel:

(3)  $S \rightarrow aAc$      $A \rightarrow bbA \mid b$     ... ist nicht  $LR(0)$ , aber  $LR(1)$ :

Für  $S \xrightarrow{*}_R \alpha X w \rightarrow \alpha \beta w$  mit  $\{y\} = \text{First}_k(w)$  ist  $\alpha \underline{\beta} y$  von einer der Formen:

$$ab^{2n} \underline{b}c, ab^{2n} \underline{bbA}c, \underline{aAc}$$

(4)  $S \rightarrow aAc$      $A \rightarrow bAb \mid b$     ... ist nicht  $LR(k)$  für jedes  $k \geq 0$ :

Betrachte einfach die Rechtsableitungen:

$$S \xrightarrow{*}_R ab^n Ab^n c \rightarrow ab^n \underline{b}b^n c$$

# LR(k)-Items

Sei  $k > 0$ .

**Idee:** Wir staten Items mit  $k$ -Vorausschau aus

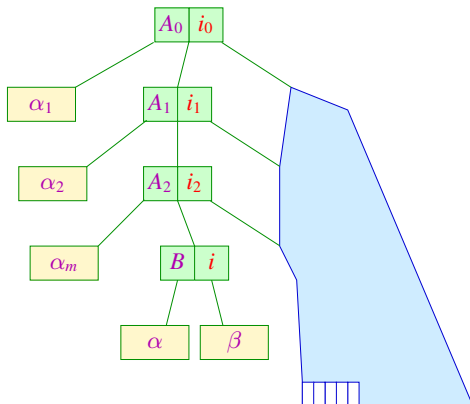
Ein  $LR(k)$ -Item ist dann ein Paar:

$$[B \rightarrow \alpha \bullet \beta, x], \quad x \in \text{Follow}_k(B)$$

Dieses Item ist gültig für  $\gamma \alpha$  falls:

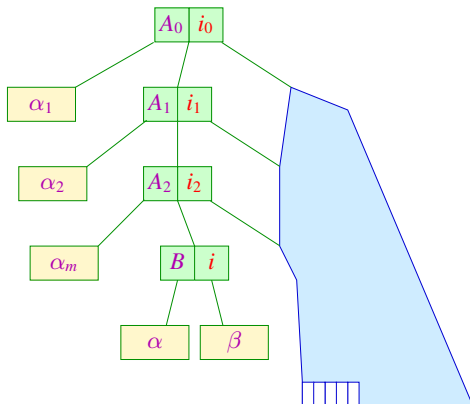
$$S \xrightarrow{*}_R \gamma B w \quad \text{mit} \quad \{x\} = \text{First}_k(w)$$

# LR(k)-Items



... wobei  $\alpha_1 \dots \alpha_m = \gamma$

# LR(k)-Items



... wobei  $\alpha_1 \dots \alpha_m = \gamma$

Die Menge der gültigen  $LR(k)$ -Items für zuverlässige Präfixe berechnen wir wieder mithilfe eines endlichen Automaten

# Der charakteristische LR(k)-Automat

Der Automat  $c(G, k)$  :

**Zustände:**  $LR(k)$ -Items

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

**Endzustände:**  $\{[B \rightarrow \gamma \bullet, x] \mid B \rightarrow \gamma \in P, x \in \text{Follow}_k(B)\}$

**Übergänge:**

(1)  $([A \rightarrow \alpha \bullet X \beta, x], X, [A \rightarrow \alpha X \bullet \beta, x]), \quad X \in (N \cup T)$

(2)  $([A \rightarrow \alpha \bullet B \beta, x], \epsilon, [B \rightarrow \bullet \gamma, x']),$   
 $A \rightarrow \alpha B \beta, \quad B \rightarrow \gamma \in P, x' \in \text{First}_k(\beta) \odot \{x\};$

# Der charakteristische LR(k)-Automat

Der Automat  $c(G, k)$  :

**Zustände:**  $LR(k)$ -Items

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

**Endzustände:**  $\{[B \rightarrow \gamma \bullet, x] \mid B \rightarrow \gamma \in P, x \in \text{Follow}_k(B)\}$

**Übergänge:**

(1)  $([A \rightarrow \alpha \bullet X \beta, x], X, [A \rightarrow \alpha X \bullet \beta, x]), \quad X \in (N \cup T)$

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

$A \rightarrow \alpha B \beta, \quad B \rightarrow \gamma \in P, x' \in \text{First}_k(\beta) \odot \{x\};$

Dieser Automat arbeitet wie  $c(G)$  — verwaltet aber zusätzlich ein  $k$ -Präfix aus dem  $\text{Follow}_k$  der linken Seiten.



## Der kanonische LR(k)-Automat

Den kanonischen  $LR(k)$ -Automaten  $LR(G, k)$  erhält man aus  $c(G, k)$ , indem man nach jedem Übergang beliebig viele  $\epsilon$  liest und dann den Automaten **deterministisch** macht ...

# Der kanonische LR(k)-Automat

Den kanonischen  $LR(k)$ -Automaten  $LR(G, k)$  erhält man aus  $c(G, k)$ , indem man nach jedem Übergang beliebig viele  $\epsilon$  liest und dann den Automaten **deterministisch** macht ...

Man kann ihn aber auch **direkt** aus der Grammatik konstruieren

Wie bei  $LR(0)$  benötigt man eine Hilfsfunktion:

$$\delta_{\epsilon}^*(q) = q \cup \{ [B \rightarrow \bullet \gamma, x] \mid \exists [A \rightarrow \alpha \bullet B' \beta', x'] \in q, \\ \beta \in (N \cup T)^* : B' \xrightarrow{*} B \beta \} \wedge \\ x \in \text{First}_k(\beta \beta') \odot \{x'\} \}$$

# Der kanonische LR(k)-Automat

Den kanonischen  $LR(k)$ -Automaten  $LR(G, k)$  erhält man aus  $c(G, k)$ , indem man nach jedem Übergang beliebig viele  $\epsilon$  liest und dann den Automaten **deterministisch** macht ...

Man kann ihn aber auch **direkt** aus der Grammatik konstruieren

Wie bei  $LR(0)$  benötigt man eine Hilfsfunktion:

$$\delta_{\epsilon}^*(q) = q \cup \{ [B \rightarrow \bullet \gamma, x] \mid \exists [A \rightarrow \alpha \bullet B' \beta', x'] \in q, \\ \beta \in (N \cup T)^* : B' \xrightarrow{*} B \beta \} \wedge \\ x \in \text{First}_k(\beta \beta') \odot \{x'\} \}$$

Dann definiert man:

**Zustände:** Mengen von  $LR(k)$ -Items;

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

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

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

# Der kanonische LR(k)-Automat

im Beispiel:

$$\begin{aligned}
 q_0 &= \left\{ \begin{array}{l} [S' \rightarrow \bullet E \quad ], \\ [E \rightarrow \bullet E + T \quad ], \\ [E \rightarrow \bullet T \quad ], \\ [T \rightarrow \bullet T * F \quad ], \\ [T \rightarrow \bullet F \quad ], \\ [F \rightarrow \bullet ( E ) \quad ], \\ [F \rightarrow \bullet \text{int} \quad ] \end{array} \right\}, & q_3 &= \delta(q_0, F) = \left\{ [T \rightarrow F \bullet \quad ] \right\} \\
 q_1 &= \delta(q_0, E) = \left\{ \begin{array}{l} [S' \rightarrow E \bullet \quad ], \\ [E \rightarrow E \bullet + T \quad ] \end{array} \right\}, & q_4 &= \delta(q_0, \text{int}) = \left\{ [F \rightarrow \text{int} \bullet \quad ] \right\} \\
 q_2 &= \delta(q_0, T) = \left\{ \begin{array}{l} [E \rightarrow T \bullet \quad ], \\ [T \rightarrow T \bullet * F \quad ] \end{array} \right\}, & q_5 &= \delta(q_0, () = \left\{ \begin{array}{l} [F \rightarrow ( \bullet E ) \quad ], \\ [E \rightarrow \bullet E + T \quad ], \\ [E \rightarrow \bullet T \quad ], \\ [T \rightarrow \bullet T * F \quad ], \\ [T \rightarrow \bullet F \quad ], \\ [F \rightarrow \bullet ( E ) \quad ], \\ [F \rightarrow \bullet \text{int} \quad ] \end{array} \right\}
 \end{aligned}$$

# Der kanonische LR(k)-Automat

im Beispiel:

$$\begin{aligned}
 q_0 &= \{ [S' \rightarrow \bullet E, \{\epsilon\}], \\
 &\quad [E \rightarrow \bullet E + T, \{\epsilon, +\}], \\
 &\quad [E \rightarrow \bullet T, \{\epsilon, +\}], \\
 &\quad [T \rightarrow \bullet T * F, \{\epsilon, +, *\}], \\
 &\quad [T \rightarrow \bullet F, \{\epsilon, +, *\}], \\
 &\quad [F \rightarrow \bullet ( E ), \{\epsilon, +, *\}], \\
 &\quad [F \rightarrow \bullet \text{int}, \{\epsilon, +, *\}] \} \\
 \\
 q_1 &= \delta(q_0, E) = \{ [S' \rightarrow E \bullet], \\
 &\quad [E \rightarrow E \bullet + T] \} \\
 \\
 q_2 &= \delta(q_0, T) = \{ [E \rightarrow T \bullet], \\
 &\quad [T \rightarrow T \bullet * F] \} \\
 \\
 q_3 &= \delta(q_0, F) = \{ [T \rightarrow F \bullet] \} \\
 \\
 q_4 &= \delta(q_0, \text{int}) = \{ [F \rightarrow \text{int} \bullet] \} \\
 \\
 q_5 &= \delta(q_0, () = \{ [F \rightarrow ( \bullet E ), \\
 &\quad [E \rightarrow \bullet E + T ], \\
 &\quad [E \rightarrow \bullet T ], \\
 &\quad [T \rightarrow \bullet T * F ], \\
 &\quad [T \rightarrow \bullet F ], \\
 &\quad [F \rightarrow \bullet ( E ), \\
 &\quad [F \rightarrow \bullet \text{int} ] \}
 \end{aligned}$$

# Der kanonische LR(k)-Automat

im Beispiel:

$$\begin{aligned}
 q_0 &= \{ [S' \rightarrow \bullet E, \{\epsilon\}], \\
 &\quad [E \rightarrow \bullet E + T, \{\epsilon, +\}], \\
 &\quad [E \rightarrow \bullet T, \{\epsilon, +\}], \\
 &\quad [T \rightarrow \bullet T * F, \{\epsilon, +, *\}], \\
 &\quad [T \rightarrow \bullet F, \{\epsilon, +, *\}], \\
 &\quad [F \rightarrow \bullet ( E ), \{\epsilon, +, *\}], \\
 &\quad [F \rightarrow \bullet \text{int}, \{\epsilon, +, *\}] \} \\
 q_1 &= \delta(q_0, E) = \{ [S' \rightarrow E \bullet, \{\epsilon\}], \\
 &\quad [E \rightarrow E \bullet + T, \{\epsilon, +\}] \} \\
 q_2 &= \delta(q_0, T) = \{ [E \rightarrow T \bullet, \{\epsilon, +\}], \\
 &\quad [T \rightarrow T \bullet * F, \{\epsilon, +, *\}] \} \\
 q_3 &= \delta(q_0, F) = \{ [T \rightarrow F \bullet, \{\epsilon, +, *\}] \} \\
 q_4 &= \delta(q_0, \text{int}) = \{ [F \rightarrow \text{int} \bullet, \{\epsilon, +, *\}] \} \\
 q_5 &= \delta(q_0, () = \{ [F \rightarrow ( \bullet E ) ], \\
 &\quad [E \rightarrow \bullet E + T ], \\
 &\quad [E \rightarrow \bullet T ], \\
 &\quad [T \rightarrow \bullet T * F ], \\
 &\quad [T \rightarrow \bullet F ], \\
 &\quad [F \rightarrow \bullet ( E ) ], \\
 &\quad [F \rightarrow \bullet \text{int} ] \}
 \end{aligned}$$

# Der kanonische LR(k)-Automat

im Beispiel:

$$\begin{aligned} q_0 &= \{ [S' \rightarrow \bullet E, \{\epsilon\}], \\ &\quad [E \rightarrow \bullet E + T, \{\epsilon, +\}], \\ &\quad [E \rightarrow \bullet T, \{\epsilon, +\}], \\ &\quad [T \rightarrow \bullet T * F, \{\epsilon, +, *\}], \\ &\quad [T \rightarrow \bullet F, \{\epsilon, +, *\}], \\ &\quad [F \rightarrow \bullet (E), \{\epsilon, +, *\}], \\ &\quad [F \rightarrow \bullet \text{int}, \{\epsilon, +, *\}] \} \\ q_1 &= \delta(q_0, E) = \{ [S' \rightarrow E \bullet, \{\epsilon\}], \\ &\quad [E \rightarrow E \bullet + T, \{\epsilon, +\}] \} \\ q_2 &= \delta(q_0, T) = \{ [E \rightarrow T \bullet, \{\epsilon, +\}], \\ &\quad [T \rightarrow T \bullet * F, \{\epsilon, +, *\}] \} \\ q_3 &= \delta(q_0, F) = \{ [T \rightarrow F \bullet, \{\epsilon, +, *\}] \} \\ q_4 &= \delta(q_0, \text{int}) = \{ [F \rightarrow \text{int} \bullet, \{\epsilon, +, *\}] \} \\ q_5 &= \delta(q_0, () = \{ [F \rightarrow ( \bullet E), \{\epsilon, +, *\}], \\ &\quad [E \rightarrow \bullet E + T, \{ \}, + \}], \\ &\quad [E \rightarrow \bullet T, \{ \}, + \}], \\ &\quad [T \rightarrow \bullet T * F, \{ \}, +, * \}], \\ &\quad [T \rightarrow \bullet F, \{ \}, +, * \}], \\ &\quad [F \rightarrow \bullet (E), \{ \}, +, * \}], \\ &\quad [F \rightarrow \bullet \text{int}, \{ \}, +, * \}] \} \end{aligned}$$

# Der kanonische LR(k)-Automat

im Beispiel:

$$\begin{array}{l}
 q'_5 = \delta(q_5, () = \{ [F \rightarrow (\bullet E) \quad ], q_7 = \delta(q_2, *) = \{ [T \rightarrow T * \bullet F \quad ], \\
 [E \rightarrow \bullet E + T \quad ], \\
 [E \rightarrow \bullet T \quad ], \\
 [T \rightarrow \bullet T * F \quad ], \\
 [T \rightarrow \bullet F \quad ], \\
 [F \rightarrow \bullet (E) \quad ], \\
 [F \rightarrow \bullet \text{int} \quad ] \} \\
 \\
 q_6 = \delta(q_1, +) = \{ [E \rightarrow E + \bullet T \quad ], q_8 = \delta(q_5, E) = \{ [F \rightarrow (E \bullet) \quad ] \\
 [T \rightarrow \bullet T * F \quad ], \\
 [T \rightarrow \bullet F \quad ], \\
 [F \rightarrow \bullet (E) \quad ], \\
 [F \rightarrow \bullet \text{int} \quad ] \} \\
 \\
 q_9 = \delta(q_6, T) = \{ [E \rightarrow E + T \bullet \quad ], \\
 [T \rightarrow T \bullet * F \quad ] \} \\
 \\
 q_{10} = \delta(q_7, F) = \{ [T \rightarrow T * F \bullet \quad ] \} \\
 \\
 q_{11} = \delta(q_8, ) = \{ [F \rightarrow (E) \bullet \quad ] \}
 \end{array}$$



# Der kanonische LR(k)-Automat

im Beispiel:

$$\begin{array}{l}
 q'_5 = \delta(q_5, () = \{ [F \rightarrow (\bullet E), \{ \}, +, *], \quad q_7 = \delta(q_2, *) = \{ [T \rightarrow T * \bullet F \quad ], \\
 [E \rightarrow \bullet E + T, \{ \}, +], \quad [F \rightarrow \bullet (E) \quad ], \\
 [E \rightarrow \bullet T, \{ \}, +], \quad [F \rightarrow \bullet \text{int} \quad ] \} \\
 [T \rightarrow \bullet T * F, \{ \}, +, *], \quad q_8 = \delta(q_5, E) = \{ [F \rightarrow (E \bullet) \quad ] \} \\
 [T \rightarrow \bullet F, \{ \}, +, *], \quad [E \rightarrow E \bullet + T \quad ] \} \\
 [F \rightarrow \bullet (E), \{ \}, +, *], \quad q_9 = \delta(q_6, T) = \{ [E \rightarrow E + T \bullet \quad ], \\
 [F \rightarrow \bullet \text{int}, \{ \}, +, *] \} \quad [T \rightarrow T \bullet * F \quad ] \} \\
 \\
 q_6 = \delta(q_1, +) = \{ [E \rightarrow E + \bullet T \quad ], \quad q_{10} = \delta(q_7, F) = \{ [T \rightarrow T * F \bullet \quad ] \} \\
 [T \rightarrow \bullet T * F \quad ], \quad [F \rightarrow \bullet (E) \quad ], \\
 [T \rightarrow \bullet F \quad ], \quad [F \rightarrow \bullet \text{int} \quad ] \} \quad q_{11} = \delta(q_8, ) = \{ [F \rightarrow (E) \bullet \quad ] \} \\
 [F \rightarrow \bullet (E) \quad ], \\
 [F \rightarrow \bullet \text{int} \quad ] \}
 \end{array}$$

# Der kanonische LR(k)-Automat

im Beispiel:

$$\begin{aligned}
 q'_5 &= \delta(q_5, () = \{ [F \rightarrow (\bullet E), \{ \}, +, *], & q_7 &= \delta(q_2, *) = \{ [T \rightarrow T * \bullet F & ], \\
 & [E \rightarrow \bullet E + T, \{ \}, +], & & [F \rightarrow \bullet (E) & ], \\
 & [E \rightarrow \bullet T, \{ \}, +], & & [F \rightarrow \bullet \text{int} & ] \} \\
 & [T \rightarrow \bullet T * F, \{ \}, +, *], & & & \\
 & [T \rightarrow \bullet F, \{ \}, +, *], & q_8 &= \delta(q_5, E) = \{ [F \rightarrow (E \bullet) & ] \} \\
 & [F \rightarrow \bullet (E), \{ \}, +, *], & & [E \rightarrow E \bullet + T & ] \} \\
 & [F \rightarrow \bullet \text{int}, \{ \}, +, *] \} & & & \\
 q_6 &= \delta(q_1, +) = \{ [E \rightarrow E + \bullet T, \{ \epsilon, + \}, & q_9 &= \delta(q_6, T) = \{ [E \rightarrow E + T \bullet & ], \\
 & [T \rightarrow \bullet T * F, \{ \epsilon, +, * \}, & & [T \rightarrow T \bullet * F & ] \} \\
 & [T \rightarrow \bullet F, \{ \epsilon, +, * \}, & q_{10} &= \delta(q_7, F) = \{ [T \rightarrow T * F \bullet & ] \} \\
 & [F \rightarrow \bullet (E), \{ \epsilon, +, * \}, & & & \\
 & [F \rightarrow \bullet \text{int}, \{ \epsilon, +, * \} \} & q_{11} &= \delta(q_8, ) = \{ [F \rightarrow (E) \bullet & ] \}
 \end{aligned}$$

# Der kanonische LR(k)-Automat

im Beispiel:

$$\begin{array}{l}
 q'_5 = \delta(q_5, () = \{ [F \rightarrow (\bullet E), \{ \}, +, *], \quad q_7 = \delta(q_2, *) = \{ [T \rightarrow T * \bullet F, \{ \epsilon, +, * \}], \\
 [E \rightarrow \bullet E + T, \{ \}, +], \quad [F \rightarrow \bullet (E), \{ \epsilon, +, * \}], \\
 [E \rightarrow \bullet T, \{ \}, +], \quad [F \rightarrow \bullet \text{int}, \{ \epsilon, +, * \}] \\
 [T \rightarrow \bullet T * F, \{ \}, +, *], \\
 [T \rightarrow \bullet F, \{ \}, +, *], \quad q_8 = \delta(q_5, E) = \{ [F \rightarrow (E \bullet), \{ \epsilon, +, * \}], \\
 [F \rightarrow \bullet (E), \{ \}, +, *], \quad [E \rightarrow E \bullet + T, \{ \}, +] \\
 [F \rightarrow \bullet \text{int}, \{ \}, +, *] \} \\
 q_6 = \delta(q_1, +) = \{ [E \rightarrow E + \bullet T, \{ \epsilon, + \}], \quad q_9 = \delta(q_6, T) = \{ [E \rightarrow E + T \bullet, \{ \epsilon, + \}], \\
 [T \rightarrow \bullet T * F, \{ \epsilon, +, * \}], \quad [T \rightarrow T \bullet * F, \{ \epsilon, +, * \}] \\
 [T \rightarrow \bullet F, \{ \epsilon, +, * \}], \quad q_{10} = \delta(q_7, F) = \{ [T \rightarrow T * F \bullet, \{ \epsilon, +, * \}] \\
 [F \rightarrow \bullet (E), \{ \epsilon, +, * \}], \\
 [F \rightarrow \bullet \text{int}, \{ \epsilon, +, * \}] \} \quad q_{11} = \delta(q_8, ) = \{ [F \rightarrow (E) \bullet, \{ \epsilon, +, * \}] \}
 \end{array}$$

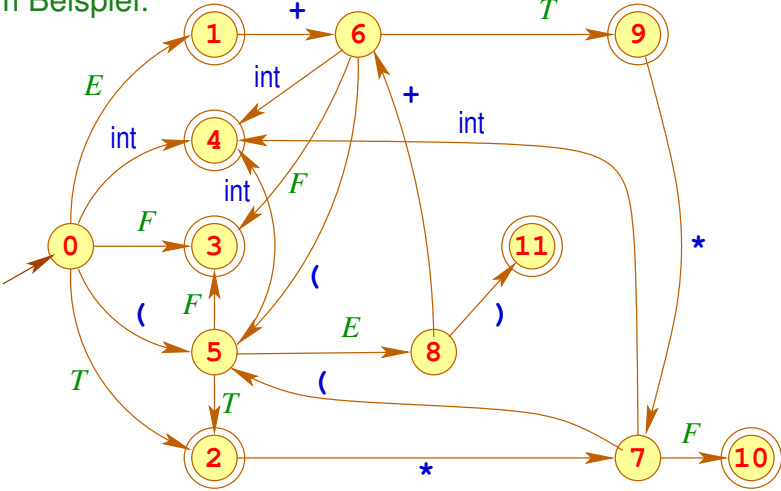
# Der kanonische LR(k)-Automat

im Beispiel:

$$\begin{aligned}q'_2 &= \delta(q'_5, T) = \left\{ \begin{array}{l} [E \rightarrow T \bullet, \{ \}, +], \\ [T \rightarrow T \bullet * F, \{ \}, +, *] \end{array} \right\} & q'_7 &= \delta(q_9, *) = \left\{ \begin{array}{l} [T \rightarrow T * \bullet F, \{ \}, +, *], \\ [F \rightarrow \bullet (E), \{ \}, +, *], \\ [F \rightarrow \bullet \text{int}, \{ \}, +, *] \end{array} \right\} \\q'_3 &= \delta(q'_5, F) = \{ [F \rightarrow F \bullet, \{ \}, +, *] \} & q'_8 &= \delta(q'_5, E) = \left\{ \begin{array}{l} [F \rightarrow (E \bullet), \{ \}, +, *], \\ [E \rightarrow E \bullet + T, \{ \}, +, *] \end{array} \right\} \\q'_4 &= \delta(q'_5, \text{int}) = \{ [F \rightarrow \text{int} \bullet, \{ \}, +, *] \} & q'_9 &= \delta(q'_6, T) = \left\{ \begin{array}{l} [E \rightarrow E + T \bullet, \{ \}, +], \\ [T \rightarrow \bullet T * F, \{ \}, +, *], \\ [T \rightarrow \bullet F, \{ \}, +, *], \\ [F \rightarrow \bullet (E), \{ \}, +, *], \\ [F \rightarrow \bullet \text{int}, \{ \}, +, *] \end{array} \right\} \\q'_6 &= \delta(q_8, +) = \left\{ \begin{array}{l} [E \rightarrow E + \bullet T, \{ \}, +], \\ [T \rightarrow T \bullet * F, \{ \}, +, *] \end{array} \right\} & q'_{10} &= \delta(q'_7, F) = \{ [T \rightarrow T * F \bullet, \{ \}, +, *] \} \\ & & q'_{11} &= \delta(q'_8, ) = \{ [F \rightarrow (E) \bullet, \{ \}, +, *] \} \end{aligned}$$

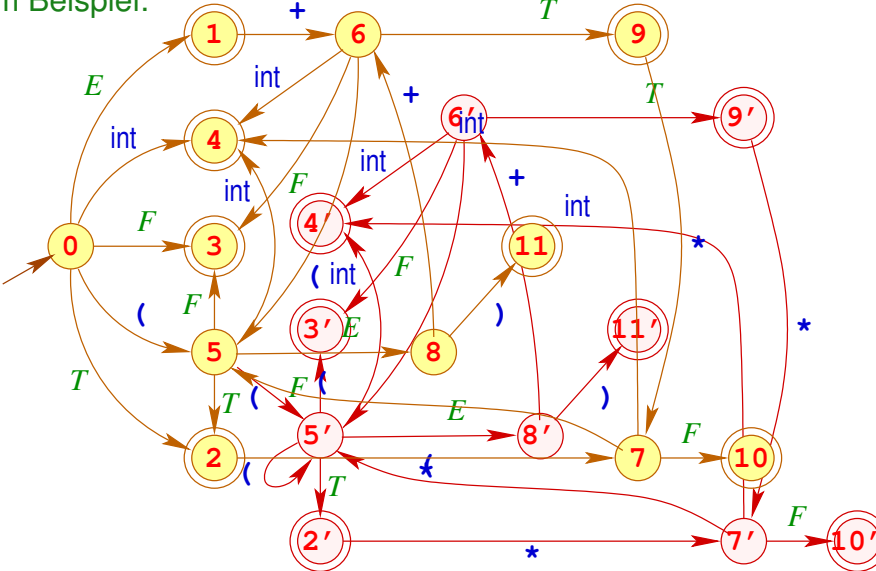
# Der kanonische LR(k)-Automat

im Beispiel:



# Der kanonische LR(k)-Automat

im Beispiel:



# Der kanonische LR(k)-Automat

## Diskussion:

- Im Beispiel hat sich die Anzahl der Zustände fast verdoppelt  
... und es kann noch schlimmer kommen
- Die Konflikte in den Zuständen  $q_1, q_2, q_9$  sind nun aufgelöst !  
z.B. haben wir für:

$$q_9 = \left\{ \begin{array}{l} [E \rightarrow E + T \bullet, \{\epsilon, +\}], \\ [T \rightarrow T \bullet * F, \{\epsilon, +, *\}] \end{array} \right\}$$

mit:

$$\{\epsilon, +\} \cap (\text{First}_1(*F) \odot \{\epsilon, +, *\}) = \{\epsilon, +\} \cap \{*\} = \emptyset$$

# Der kanonische LR(k)-Automat

**Allgemein:** Wir identifizieren zwei Konflikte:

## Reduce-Reduce-Konflikt:

$$[A \rightarrow \gamma \bullet, x], [A' \rightarrow \gamma' \bullet, x] \in q \text{ mit } A \neq A' \vee \gamma \neq \gamma'$$

## Shift-Reduce-Konflikt:

$$[A \rightarrow \gamma \bullet, x], [A' \rightarrow \alpha \bullet a \beta, y] \in q \text{ mit } a \in T \text{ und} \\ x \in \{a\} \odot \text{First}_k(\beta) \odot \{y\} .$$

für einen Zustand  $q \in Q$  .

Solche Zustände nennen wir jetzt **LR(k)-ungeeignet**



## Spezielle LR(k)-Teilklassen

### Satz:

Eine reduzierte kontextfreie Grammatik  $G$  ist genau dann  $LR(k)$  wenn der kanonische  $LR(k)$ -Automat  $LR(G, k)$  keine  $LR(k)$ -ungeeigneten Zustände besitzt.

# Spezielle LR(k)-Teilklassen

## Satz:

Eine reduzierte kontextfreie Grammatik  $G$  ist genau dann  $LR(k)$  wenn der kanonische  $LR(k)$ -Automat  $LR(G, k)$  keine  $LR(k)$ -ungeeigneten Zustände besitzt.

## Diskussion:

- Unser Beispiel ist offenbar  $LR(1)$
- Im Allgemeinen hat der kanonische  $LR(k)$ -Automat sehr viel mehr Zustände als  $LR(G) = LR(G, 0)$
- Man betrachtet darum i.a. **Teilklassen** von  $LR(k)$ -Grammatiken, bei denen man nur  $LR(G)$  benutzt ...

# Spezielle LR(k)-Teilklassen

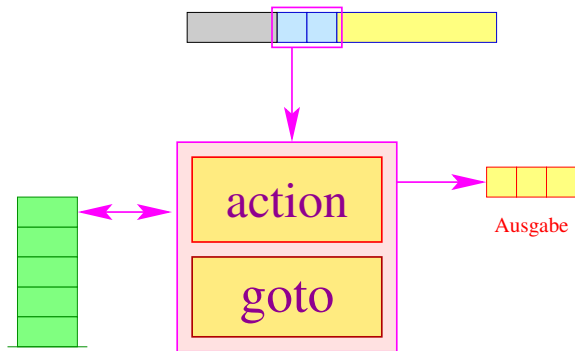
## Satz:

Eine reduzierte kontextfreie Grammatik  $G$  ist genau dann  $LR(k)$  wenn der kanonische  $LR(k)$ -Automat  $LR(G, k)$  keine  $LR(k)$ -ungeeigneten Zustände besitzt.

## Diskussion:

- Unser Beispiel ist offenbar  $LR(1)$
- Im Allgemeinen hat der kanonische  $LR(k)$ -Automat sehr viel mehr Zustände als  $LR(G) = LR(G, 0)$
- Man betrachtet darum i.a. **Teilklassen** von  $LR(k)$ -Grammatiken, bei denen man nur  $LR(G)$  benutzt ...
- Zur Konflikt-Auflösung ordnet man den Items in den Zuständen Vorausschau-Mengen zu:
  - 1 Die Zuordnung ist unabhängig vom Zustand  $\implies$  Simple  $LR(k)$
  - 2 Die Zuordnung hängt vom Zustand ab  $\implies$   $LALR(k)$

## Der $LR(k)$ -Parser:



- Die **goto**-Tabelle kodiert die Zustandsübergänge:

$$\text{goto}[q, X] = \delta(q, X) \in Q$$

- Die **action**-Tabelle beschreibt für jeden Zustand  $q$  und möglichen Look-ahead  $w$  die erforderliche Aktion.

# Der $LR(k)$ -Parser:

Mögliche Aktionen sind:

**shift** // Shift-Operation  
**reduce** ( $A \rightarrow \gamma$ ) // Reduktion mit Ausgabe  
**error** // Fehler

... im Beispiel:

$E \rightarrow E+T^0$  |  $T^1$   
 $T \rightarrow T*F^0$  |  $F^1$   
 $F \rightarrow (E)^0$  |  $\text{int}^1$

action	$\epsilon$	int	( )	+	*
$q_1$	$S', 0$				S
$q_2$	$E, 1$				S
$q'_2$			$E, 1$		S
$q_3$	$T, 1$			$T, 1$	$T, 1$
$q'_3$			$T, 1$	$T, 1$	$T, 1$
$q_4$	$F, 1$			$F, 1$	$F, 1$
$q'_4$			$F, 1$	$F, 1$	$F, 1$
$q_9$	$E, 0$			$E, 0$	S
$q'_9$			$E, 0$	$E, 0$	S
$q_{10}$	$T, 0$			$T, 0$	$T, 0$
$q'_{10}$			$T, 0$	$T, 0$	$T, 0$
$q_{11}$	$F, 0$			$F, 0$	$F, 0$
$q'_{11}$			$F, 0$	$F, 0$	$F, 0$

## simple- $LR(k)$

**Idee 1:** Benutze  $\text{Follow}_k$ -Mengen zur Konflikt-Lösung

### Reduce-Reduce-Konflikt:

Falls für  $[A \rightarrow \gamma \bullet], [A' \rightarrow \gamma' \bullet] \in q$  mit  $A \neq A' \vee \gamma \neq \gamma'$ ,

$$\text{Follow}_k(A) \cap \text{Follow}_k(A') \neq \emptyset$$

### Shift-Reduce-Konflikt:

Falls für  $[A \rightarrow \gamma \bullet], [A' \rightarrow \alpha \bullet a \beta] \in q$  mit  $a \in T$ ,

$$\text{Follow}_k(A) \cap (\{a\} \odot \text{First}_k(\beta) \odot \text{Follow}_k(A')) \neq \emptyset$$

für einen Zustand  $q \in Q$ .

Dann nennen wir den Zustand  $q$   $SLR(k)$ -ungeeignet

## simple- $LR(k)$

### Definition:

Die reduzierte Grammatik  $G$  nennen wir  $SLR(k)$  (simple  $LR(k)$ ) falls der kanonische  $LR(0)$ -Automat  $LR(G)$  keine  $SLR(k)$ -ungeeigneten Zustände enthält

## simple- $LR(k)$

### Definition:

Die reduzierte Grammatik  $G$  nennen wir  $SLR(k)$  (simple  $LR(k)$ ) falls der kanonische  $LR(0)$ -Automat  $LR(G)$  keine  $SLR(k)$ -ungeeigneten Zustände enthält

### ... im Beispiel:

Bei unserer Beispiel-Grammatik treten Konflikte möglicherweise in den Zuständen  $q_1, q_2, q_9$  auf:

$$q_1 = \left\{ \begin{array}{l} [S' \rightarrow E \bullet], \\ [E \rightarrow E \bullet + T] \end{array} \right\} \quad \text{Follow}_1(S') \cap \{+\} \odot \{\dots\} = \{\epsilon\} \cap \{+\} = \emptyset$$



## simple- $LR(k)$

### Definition:

Die reduzierte Grammatik  $G$  nennen wir  $SLR(k)$  (simple  $LR(k)$ ) falls der kanonische  $LR(0)$ -Automat  $LR(G)$  keine  $SLR(k)$ -ungeeigneten Zustände enthält

### ... im Beispiel:

Bei unserer Beispiel-Grammatik treten Konflikte möglicherweise in den Zuständen  $q_1, q_2, q_9$  auf:

$$q_1 = \left\{ \begin{array}{l} [S' \rightarrow E \bullet], \\ [E \rightarrow E \bullet + T] \end{array} \right\} \quad \text{Follow}_1(S') \cap \{+\} \odot \{\dots\} = \{\epsilon\} \cap \{+\} = \emptyset$$

$$q_2 = \left\{ \begin{array}{l} [E \rightarrow T \bullet], \\ [T \rightarrow T \bullet * F] \end{array} \right\} \quad \text{Follow}_1(E) \cap \{*\} \odot \{\dots\} = \{\epsilon, +, )\} \cap \{*\} = \emptyset$$

$$q_9 = \left\{ \begin{array}{l} [E \rightarrow E + T \bullet], \\ [T \rightarrow T \bullet * F] \end{array} \right\} \quad \text{Follow}_1(E) \cap \{*\} \odot \{\dots\} = \{\epsilon, +, )\} \cap \{*\} = \emptyset$$

## LALR(k)

Idee 2: Berechne für jeden Zustand  $q$  Follow-Mengen

Für  $[A \rightarrow \alpha \bullet \beta] \in q$  definieren wir:

$$\begin{aligned}\Lambda_k(q, [A \rightarrow \alpha \bullet \beta]) &= \{\text{First}_k(w) \mid S' \xrightarrow{*}_R \gamma A w \wedge \\ &\quad \delta(q_0, \gamma \alpha) = q\} \\ // &\subseteq \text{Follow}_k(A)\end{aligned}$$

**Idee 2:** Berechne für jeden Zustand  $q$  Follow-Mengen

Für  $[A \rightarrow \alpha \bullet \beta] \in q$  definieren wir:

$$\begin{aligned}\Lambda_k(q, [A \rightarrow \alpha \bullet \beta]) &= \{\text{First}_k(w) \mid S' \xrightarrow{*}_R \gamma A w \wedge \\ &\quad \delta(q_0, \gamma \alpha) = q\} \\ // &\subseteq \text{Follow}_k(A)\end{aligned}$$

**Reduce-Reduce-Konflikt:**

$[A \rightarrow \gamma \bullet], [A' \rightarrow \gamma' \bullet] \in q$  mit  $A \neq A' \vee \gamma \neq \gamma'$  wobei:

$$\Lambda_k(q, [A \rightarrow \gamma \bullet]) \cap \Lambda_k(q, [A' \rightarrow \gamma' \bullet]) \neq \emptyset$$

**Idee 2:** Berechne für jeden Zustand  $q$  Follow-Mengen

Für  $[A \rightarrow \alpha \bullet \beta] \in q$  definieren wir:

$$\begin{aligned}\Lambda_k(q, [A \rightarrow \alpha \bullet \beta]) &= \{\text{First}_k(w) \mid S' \xrightarrow{*}_R \gamma A w \wedge \\ &\quad \delta(q_0, \gamma \alpha) = q\} \\ // &\subseteq \text{Follow}_k(A)\end{aligned}$$

**Reduce-Reduce-Konflikt:**

$[A \rightarrow \gamma \bullet], [A' \rightarrow \gamma' \bullet] \in q$  mit  $A \neq A' \vee \gamma \neq \gamma'$  wobei:

$$\Lambda_k(q, [A \rightarrow \gamma \bullet]) \cap \Lambda_k(q, [A' \rightarrow \gamma' \bullet]) \neq \emptyset$$

**Shift-Reduce-Konflikt:**

$[A \rightarrow \gamma \bullet], [A' \rightarrow \alpha \bullet a \beta] \in q$  mit  $a \in T$  wobei:

$$\Lambda_k(q, [A \rightarrow \gamma \bullet]) \cap (\{a\} \odot \text{First}_k(\beta) \odot \Lambda_k(q, [A' \rightarrow \alpha \bullet a \beta])) \neq \emptyset$$

Solche Zustände nennen wir jetzt **LALR(k)-ungeeignet**

## $LALR(k)$

### Definition:

Die reduzierte Grammatik  $G$  nennen wir  $LALR(k)$ , falls der kanonische  $LR(0)$ -Automat  $LR(G)$  keine  $LALR(k)$ -ungeeigneten Zustände enthält

Bevor wir Beispiele betrachten, überlegen wir erst, wie die Mengen  $\Lambda_k(q, [A \rightarrow \alpha \bullet \beta])$  berechnet werden können

# LALR(k)

## Definition:

Die reduzierte Grammatik  $G$  nennen wir  $LALR(k)$ , falls der kanonische  $LR(0)$ -Automat  $LR(G)$  keine  $LALR(k)$ -ungeeigneten Zustände enthält

Bevor wir Beispiele betrachten, überlegen wir erst, wie die Mengen  $\Lambda_k(q, [A \rightarrow \alpha \bullet \beta])$  berechnet werden können

**Idee:** Stelle ein Ungleichungssystem für  $\Lambda_k$  auf !

$$\begin{array}{lll} \Lambda_k(q_0, [S' \rightarrow \bullet S]) & \supseteq & \{\epsilon\} \\ \Lambda_k(q, [A \rightarrow \alpha X \bullet \beta]) & \supseteq & \Lambda_k(p, [A \rightarrow \alpha \bullet X \beta]) \quad \text{falls } \delta(p, X) = q \\ \Lambda_k(q, [A \rightarrow \bullet \gamma]) & \supseteq & \text{First}_k(\beta) \odot \Lambda_k(q, [B \rightarrow \alpha \bullet A \beta]) \quad \text{falls } [B \rightarrow \alpha \bullet A \beta] \in q \end{array}$$

## LALR(k)

im Beispiel:

$$\begin{array}{l} S \rightarrow AbB \mid B \\ A \rightarrow a \mid bB \\ B \rightarrow A \end{array}$$

Der kanonische LR(0)-Automat hat dann die folgenden Zustände:

$$\begin{array}{ll} q_0 = \{ [S' \rightarrow \bullet S], & q_2 = \delta(q_0, a) = \{ [A \rightarrow a \bullet] \} \\ [S \rightarrow \bullet AbB], & \\ [A \rightarrow \bullet a], & q_3 = \delta(q_0, b) = \{ [A \rightarrow b \bullet B], \\ [A \rightarrow \bullet bB], & [B \rightarrow \bullet A], \\ [S \rightarrow \bullet B], & [A \rightarrow \bullet a], \\ [B \rightarrow \bullet A] \} & [A \rightarrow \bullet bB] \} \end{array}$$
$$\begin{array}{ll} q_1 = \delta(q_0, S) = \{ [S' \rightarrow S \bullet] \} & q_4 = \delta(q_0, B) = \{ [S \rightarrow B \bullet] \} \end{array}$$

# LALR(k)

im Beispiel:

$$q_5 = \delta(q_0, A) = \left\{ \begin{array}{l} [S \rightarrow A \bullet b B], \\ [B \rightarrow A \bullet] \end{array} \right\}, \quad q_8 = \delta(q_5, b) = \left\{ \begin{array}{l} [S \rightarrow A b \bullet B], \\ [B \rightarrow \bullet A], \\ [A \rightarrow \bullet a], \\ [A \rightarrow \bullet b B] \end{array} \right\}$$

$$q_6 = \delta(q_3, A) = \{[B \rightarrow A \bullet]\}$$

$$q_7 = \delta(q_3, B) = \{[A \rightarrow b B \bullet]\}$$

$$q_9 = \delta(q_8, B) = \{[S \rightarrow A b B \bullet]\}$$

**Shift-Reduce-Konflikt:**

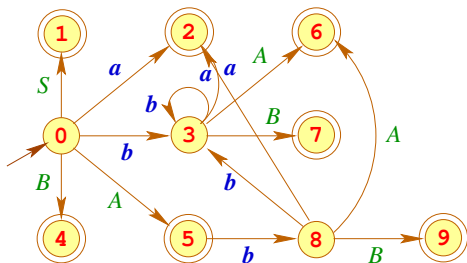
$$q_5 = \left\{ \begin{array}{l} [S \rightarrow A \bullet b B], \\ [B \rightarrow A \bullet] \end{array} \right\}$$

Dabei ist:  $\text{Follow}_1(B) \cap \{b\} \odot \{\dots\} = \{\epsilon, b\} \cap \{b\} \neq \emptyset$



# LALR(k)

im Beispiel:



Ausschnitt des Ungleichungssystems:

$$\begin{aligned} \Lambda_1(q_1, [B \rightarrow A\bullet]) &\supseteq \Lambda_1(q_0, [B \rightarrow \bullet A]) & \Lambda_1(q_0, [B \rightarrow \bullet A]) &\supseteq \Lambda_1(q_0, [S \rightarrow \bullet B]) \\ & & \Lambda_1(q_0, [S \rightarrow \bullet B]) &\supseteq \Lambda_1(q_0, [S' \rightarrow \bullet S]) \\ & & \Lambda_1(q_0, [S' \rightarrow \bullet S]) &\supseteq \{\epsilon\} \end{aligned}$$

Folglich:

$$\Lambda_1(q_5, [B \rightarrow A\bullet]) = \{\epsilon\}$$

# Spezielle Bottom-up Verfahren mit LR(G)

## Diskussion:

- Das Beispiel ist folglich **nicht**  $SLR(1)$ , aber  $LALR(1)$
- Das Beispiel ist nicht so an den Haaren herbei gezogen, wie es scheint ...
- Umbenennung:  $A \Rightarrow L$   $B \Rightarrow R$   $a \Rightarrow id$   $b \Rightarrow * / =$  liefert:

$$\begin{aligned} S &\rightarrow L = R \mid R \\ L &\rightarrow id \mid *R \\ R &\rightarrow L \end{aligned}$$

... d.h. ein Fragment der Grammatik für C-Ausdrücke

# LALR(1)

Für  $k = 1$  lassen sich die Mengen  $\Lambda_k(q, [A \rightarrow \alpha \bullet \beta])$  wieder effizient berechnen

## Das verbesserte Ungleichungssystem für $\Lambda_1$ :

$$\begin{array}{ll} \Lambda_1(q_0, [S' \rightarrow \bullet S]) & \supseteq \{\epsilon\} \\ \Lambda_1(q, [A \rightarrow \alpha X \bullet \beta]) & \supseteq \Lambda_1(p, [A \rightarrow \alpha \bullet X \beta]) \quad \text{falls } \delta(p, X) = q \\ \Lambda_1(q, [A \rightarrow \bullet \gamma]) & \supseteq F_\epsilon(X_j) \quad \text{falls } [B \rightarrow \alpha \bullet A X_1 \dots X_m] \in q \\ & \quad \text{und } \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1}) \\ \Lambda_1(q, [A \rightarrow \bullet \gamma]) & \supseteq \Lambda_1(q, [B \rightarrow \alpha \bullet A X_1 \dots X_m]) \quad \text{falls } [B \rightarrow \alpha \bullet A X_1 \dots X_m] \in q \\ & \quad \text{und } \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_m) \end{array}$$

$\implies$  wieder ein reines Vereinigungsproblem !

# Parsing Verfahren

deterministische Sprachen

= LR(1) = ... = LR(k)

LR(0)

reguläre  
Sprachen

LL(1)



LL(k)



# Parsing Verfahren

## Diskussion:

- Alle kontextfreien Sprachen, die sich mit einem deterministischen Kellerautomaten parsen lassen, können durch eine **LR(1)**-Grammatik beschrieben werden.
- Durch **LR(0)**-Grammatiken lassen sich alle **präfixfreien** deterministisch kontextfreien Sprachen beschreiben
- Die Sprachklassen zu **LL(k)**-Grammatiken bilden dagegen eine **Hierarchie** innerhalb der deterministisch kontextfreien Sprachen.
- Da zu jeder **LL(k)**-Grammatik eine **äquivalente starke LL(k)**-Grammatik konstruiert werden kann, sind letztere nicht in der Übersicht vermerkt.