

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 :-)**

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 :-)

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 auf !!!

$$\begin{aligned}
 \Lambda_k(q_0, [S' \rightarrow \bullet S]) &\supseteq \{\epsilon\} \\
 \Lambda_k(q, [A \rightarrow \alpha X \bullet \beta]) &\supseteq \Lambda_k(p, [A \rightarrow \alpha \bullet X \beta]) && \text{falls } \delta(p, X) = q \\
 \Lambda_k(q, [A \rightarrow \bullet \gamma]) &\supseteq \text{First}_k(\beta) \odot \Lambda_k(q, [B \rightarrow \alpha \bullet A \beta]) && \text{falls } [B \rightarrow \alpha \bullet A \beta] \in q
 \end{aligned}$$

Beispiel:

$$S \rightarrow AbB \mid B$$

$$A \rightarrow a \mid bB$$

$$B \rightarrow A$$

Der kanonische  $LR(0)$ -Automat hat dann die folgenden Zustände:

$$q_0 = \{ [S' \rightarrow \bullet S], [S \rightarrow \bullet AbB], [A \rightarrow \bullet a], [A \rightarrow \bullet bB], [S \rightarrow \bullet B], [B \rightarrow \bullet A] \}$$

$$q_2 = \delta(q_0, a) = \{ [A \rightarrow a\bullet] \}$$

$$q_3 = \delta(q_0, b) = \{ [A \rightarrow b\bullet B], [B \rightarrow \bullet A], [A \rightarrow \bullet a], [A \rightarrow \bullet bB] \}$$

$$q_1 = \delta(q_0, S) = \{ [S' \rightarrow S\bullet] \}$$

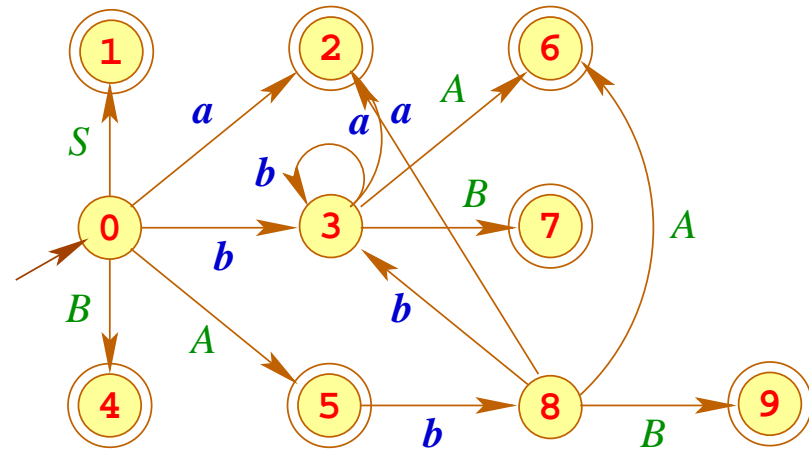
$$q_4 = \delta(q_0, B) = \{ [S \rightarrow B\bullet] \}$$

$$\begin{aligned}
q_5 &= \delta(q_0, A) = \{[S \rightarrow A \bullet b B], & q_8 &= \delta(q_5, b) = \{[S \rightarrow A b \bullet B], \\
& [B \rightarrow A \bullet]\} & & [B \rightarrow \bullet A], \\
q_6 &= \delta(q_3, A) = \{[B \rightarrow A \bullet]\} & & [A \rightarrow \bullet a], \\
q_7 &= \delta(q_3, B) = \{[A \rightarrow b B \bullet]\} & & [A \rightarrow \bullet b B]\} \\
q_9 &= \delta(q_8, B) = \{[S \rightarrow A b B \bullet]\}
\end{aligned}$$

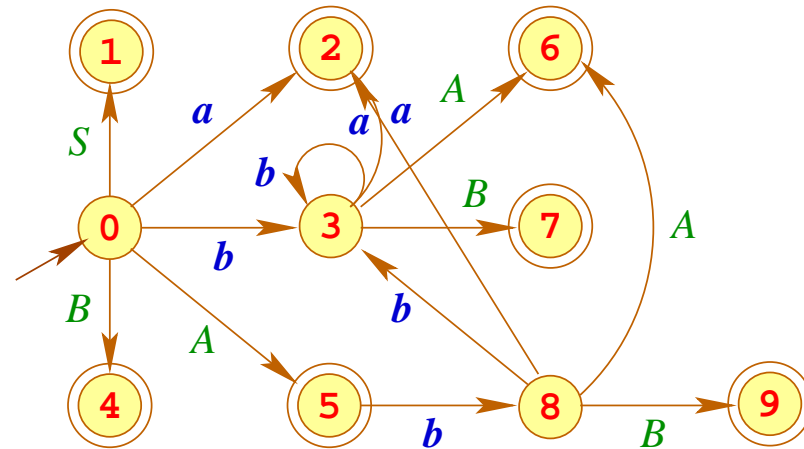
**Shift-Reduce-Konflikt:**

$$q_5 = \{[S \rightarrow A \bullet b B], \\ [B \rightarrow A \bullet]\}$$

Dabei ist:  $\text{Follow}_1(B) \cap \{b\} \odot \{\dots\} = \{\epsilon, b\} \cap \{b\} \neq \emptyset$

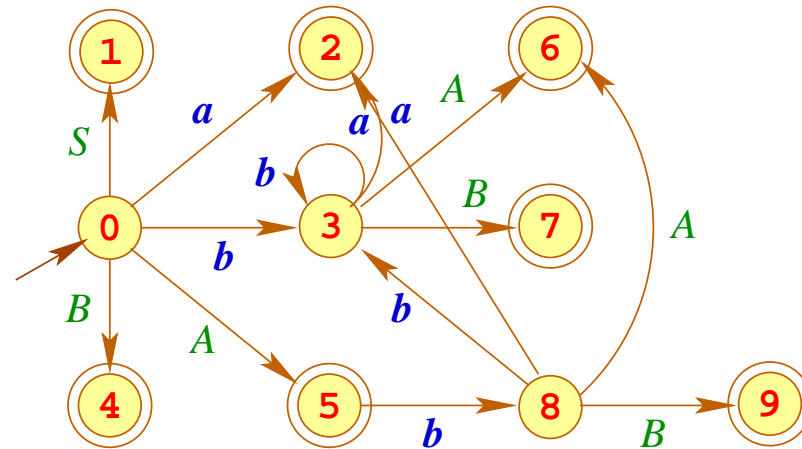






Ausschnitt des Ungleichungssystems:

$$\begin{aligned}
 \Lambda_1(q_5, [B \rightarrow A\bullet]) &\supseteq \Lambda_1(q_0, [B \rightarrow \bullet A]) & \Lambda_1(q_0, [B \rightarrow \bullet A]) &\supseteq \Lambda_1(q_0, [S \rightarrow \bullet B]) \\
 & & \Lambda_1(q_0, [S \rightarrow \bullet B]) &\supseteq \Lambda_1(q_0, [S' \rightarrow \bullet S]) \\
 & & \Lambda_1(q_0, [S' \rightarrow \bullet S]) &\supseteq \{\epsilon\}
 \end{aligned}$$



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:  $\Lambda_1(q_5, [B \rightarrow A\bullet]) = \{\epsilon\}$

## 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:

$$S \rightarrow L = R \mid R$$

$$L \rightarrow id \mid * R$$

$$R \rightarrow L$$

... d.h. ein Fragment der Grammatik für C-Ausdrücke ;-)

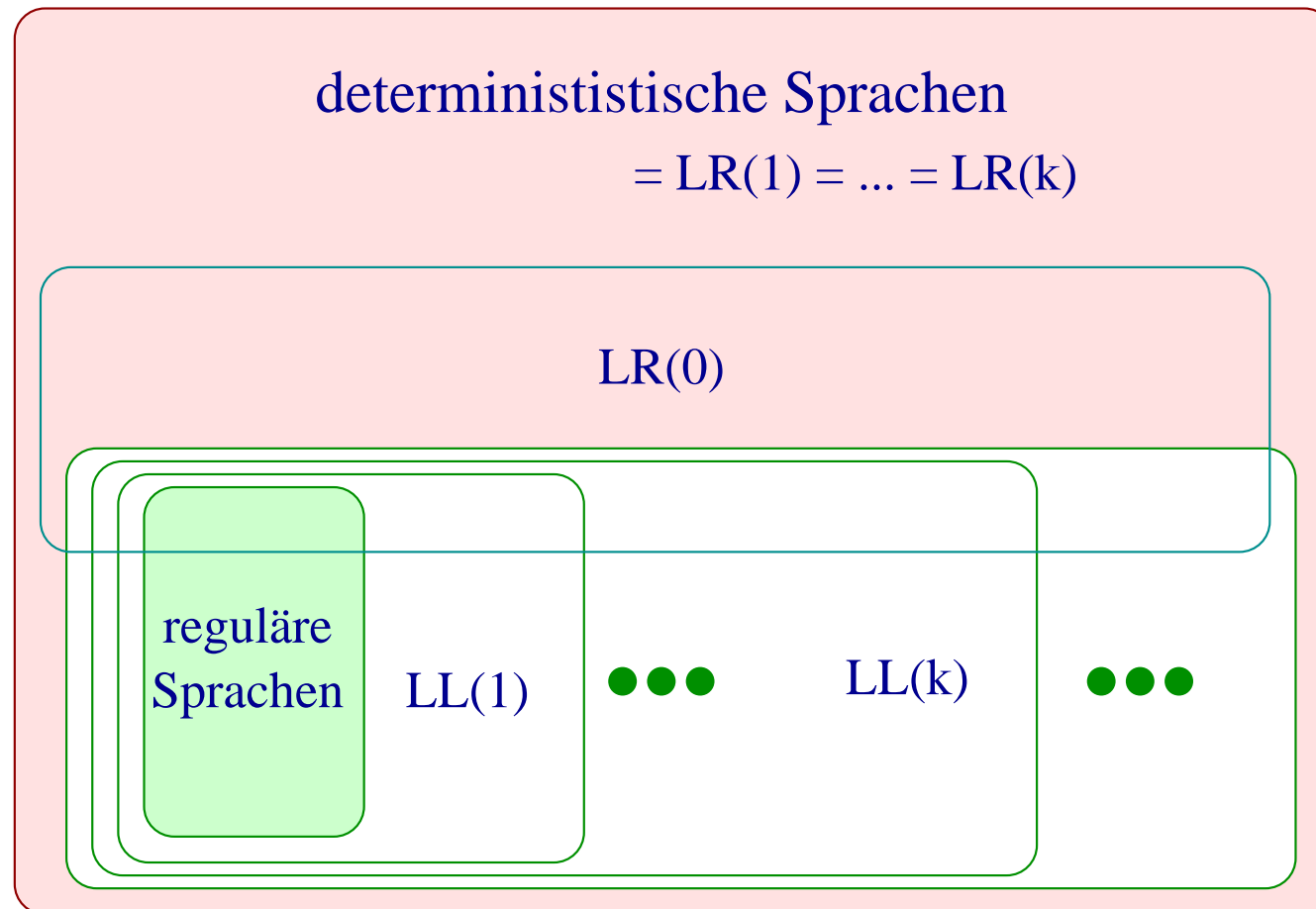
Für  $k = 1$  lassen sich die Mengen  $\Lambda_k(q, [A \rightarrow \alpha \bullet \beta])$  wieder effizient berechnen :-)

## Das verbesserte Ungleichungssystem:

$$\begin{array}{lll}
 \Lambda_1(q_0, [S' \rightarrow \bullet S]) & \supseteq & \{\epsilon\} \\
 \Lambda_1(q, [A \rightarrow \alpha X \bullet \beta]) & \supseteq & \Lambda_1(p, [A \rightarrow \alpha \bullet X \beta]) \quad \text{falls } \delta(p, X) = q \\
 \Lambda_1(q, [A \rightarrow \bullet \gamma]) & \supseteq & F_\epsilon(X_j) \quad \text{falls } [B \rightarrow \alpha \bullet A X_1 \dots X_m] \in q \\
 & & \text{und } \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_{j-1}) \\
 \Lambda_1(q, [A \rightarrow \bullet \gamma]) & \supseteq & \Lambda_1(q, [B \rightarrow \alpha \bullet A X_1 \dots X_m]) \quad \text{falls } [B \rightarrow \alpha \bullet A X_1 \dots X_m] \in q \\
 & & \text{und } \text{empty}(X_1) \wedge \dots \wedge \text{empty}(X_m)
 \end{array}$$

$\implies$  ein reines Vereinigungsproblem :-))

# Übersicht über die Sprachklassen:



## 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.

### 3 Semantische Analyse

- Lexikalisch und syntaktisch korrekte Programme können trotzdem fehlerhaft sein ;-(
- Einige von diesen Fehlern werden bereits durch die Sprachdefinition ausgeschlossen und müssen vom Compiler überprüft werden :-)
- Weitere Analysen sind erforderlich, um:
  - Bezeichner eindeutig zu machen;
  - die Typen von Variablen zu ermitteln;
  - Möglichkeiten zur Programm-Optimierung zu finden.

## 3.1 Symbol-Tabellen

Beispiel:

```
void foo() {  
    int A;  
    void fee() {  
        double A;  
        A = 0.5;  
        write(A);  
    }  
    A = 2;  
    fee();  
    write(A);  
}
```



## Diskussion:

- Innerhalb des Rumpfs von `fee` wird die Definition von `A` durch die **lokale Definition** verdeckt :-)
- Für die Code-Erzeugung benötigen wir für jede Benutzung eines Bezeichners die zugehörige **Definitionsstelle**.
- **Statische Bindung** bedeutet, dass die Definition eines Namens `A` an allen Programmpunkten innerhalb ihres gesamten Blocks **gültig** ist.
- **Sichtbar** ist sie aber nur außerhalb derjenigen Teilbereiche, in an denen eine weitere Definition von `A` gültig ist :-)

... im Beispiel:

```
void foo() {
```

```
    int A;
```

```
    void fee() {
```

```
        double A;
```

```
        A = 0.5;
```

```
        write(A);
```

```
    }
```

```
    A = 2;
```

```
    fee();
```

```
    write(A);
```

```
}
```

Kompliziertere Regeln der Sichtbarkeit gibt es in objektorientierten Programmiersprachen wie **Java ...**

## Beispiel:

```
public class Foo {  
    protected int x = 17;  
    protected int y = 5;  
    private int z = 42;  
    public int b() { return 1; }  
}  
class Fee extends Foo {  
    protected double y = .5;  
    public int b(int a) { return a; }  
}
```

## Diskussion:

- **private** Members sind nur innerhalb der aktuellen Klasse gültig :-)
- **protected** Members sind innerhalb der Klasse, in den Unterklassen sowie innerhalb des gesamten **package** gültig :-)
- Methoden **b** gleichen Namens sind stets verschieden, wenn ihre Argument-Typen verschieden sind !!!
- Bei Aufrufen einer Methode wird **dynamisch** entschieden, welche Definition gemeint ist ...

## Beispiel:

```
public class Foo {  
    protected int foo() { return 1; }  
}  
class Fee extends Foo {  
    protected int foo() { return 2; }  
    public int test(boolean b) {  
        Foo x = (b) ? new Foo() : new Fee();  
        return x.foo();  
    }  
}
```