2.7 Spezielle Bottom-up-Verfahren mit $LR(G)$

Idee 1: Benutze Follow_k-Mengen zur Konflikt-Lösung ...

Reduce-Reduce-Konflikt:

Falls für $[A \to \gamma \bullet], \; [A' \to \gamma' \bullet] \;\in\; q$ mit $A \neq A' \vee \gamma \neq \gamma'$,

$$\mathsf{Follow}_k(A) \cap \mathsf{Follow}_k(A') \neq \emptyset$$

Shift-Reduce-Konflikt:

Falls für $[A \to \gamma \bullet], \; [A' \to \alpha \bullet a\,\beta] \;\in\; q$ mit $a \in T$,

$$\mathsf{Follow}_k(A) \cap (\{a\} \odot \mathsf{First}_k(\beta) \odot \mathsf{Follow}_k(A')) \neq \emptyset$$

für einen Zustand $q \in Q$.

Dann nennen wir den Zustand q $SLR(k)$-ungeeignet :-)

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 :-)

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:

$$q_1 = \{[S' \to E\bullet], \quad [E \to E \bullet + T]\}$$

$$\mathsf{Follow}_1(S') \cap \{+\} \odot \{\ldots\} = \{\epsilon\} \cap \{+\} = \emptyset$$

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:

$$
\begin{array}{lcl}
q_1 & = & \{[S' \to E\bullet], \\
 & & \quad [E \to E \bullet + T]\}
\end{array}
\qquad
\begin{array}{lcl}
\mathsf{Follow}_1(S') \cap \{+\} \odot \{\ldots\} & = & \{\epsilon\} \cap \{+\} \\
 & = & \emptyset
\end{array}
$$

$$
\begin{array}{lcl}
q_2 & = & \{[E \to T\bullet], \\
 & & \quad [T \to T \bullet * F]\}
\end{array}
\qquad
\begin{array}{lcl}
\mathsf{Follow}_1(E) \cap \{*\} \odot \{\ldots\} & = & \{\epsilon, +,)\} \cap \{*\} \\
 & = & \emptyset
\end{array}
$$

$$
\begin{array}{lcl}
q_9 & = & \{[E \to E + T\bullet], \\
 & & \quad [T \to T \bullet * F]\}
\end{array}
\qquad
\begin{array}{lcl}
\mathsf{Follow}_1(E) \cap \{*\} \odot \{\ldots\} & = & \{\epsilon, +,)\} \cap \{*\} \\
 & = & \emptyset
\end{array}
$$

Idee 2: Berechne für jeden Zustand q Follow-Mengen :-)

Für $[A \to \alpha \bullet \beta] \in q$ definieren wir:

$$\begin{array}{rcl} \Lambda_k(q, [A \to \alpha \bullet \beta]) & = & \{\mathsf{First}_k(w) \mid S' \to_R^* \gamma\, A\, w \;\wedge \\ & & \qquad \delta(q_0, \gamma\, \alpha) = q\} \\ & // \quad \subseteq & \mathsf{Follow}_k(A) \end{array}$$

Idee 2: Berechne für jeden Zustand q Follow-Mengen :-)

Für $[A \to \alpha \bullet \beta] \in q$ definieren wir:

$$\begin{array}{rcl} \Lambda_k(q, [A \to \alpha \bullet \beta]) & = & \{\mathsf{First}_k(w) \mid S' \to_R^* \gamma\, A\, w \;\wedge \\ & & \qquad \delta(q_0, \gamma\, \alpha) = q\} \\ & // \quad \subseteq & \mathsf{Follow}_k(A) \end{array}$$

Reduce-Reduce-Konflikt:

$[A \to \gamma \bullet], \; [A' \to \gamma' \bullet] \in q$ mit $A \neq A' \vee \gamma \neq \gamma'$ wobei:

$$\Lambda_k(q, [A \to \gamma \bullet]) \cap \Lambda_k(q, [A' \to \gamma' \bullet]) \neq \emptyset$$

Idee 2: Berechne für jeden Zustand q Follow-Mengen :-)

Für $[A \to \alpha \bullet \beta] \in q$ definieren wir:

$$\begin{array}{rcl} \Lambda_k(q, [A \to \alpha \bullet \beta]) & = & \{\mathsf{First}_k(w) \mid S' \to_R^* \gamma\, A\, w \;\wedge \\ & & \qquad \delta(q_0, \gamma\, \alpha) = q\} \\ & // \quad \subseteq & \mathsf{Follow}_k(A) \end{array}$$

Reduce-Reduce-Konflikt:

$$[A \to \gamma \bullet], \; [A' \to \gamma' \bullet] \;\in\; q \quad \text{mit} \quad A \neq A' \vee \gamma \neq \gamma' \quad \text{wobei:}$$

$$\Lambda_k(q, [A \to \gamma \bullet]) \;\cap\; \Lambda_k(q, [A' \to \gamma' \bullet]) \;\neq\; \emptyset$$

Shift-Reduce-Konflikt:

$$[A \to \gamma \bullet], \; [A' \to \alpha \bullet a\, \beta] \;\in\; q \quad \text{mit} \quad a \in T \quad \text{wobei:}$$

$$\Lambda_k(q, [A \to \gamma \bullet]) \;\cap\; (\{a\} \odot \mathsf{First}_k(\beta) \odot \Lambda_k(q, [A' \to \alpha \bullet a\, \beta])) \;\neq\; \emptyset$$

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 $\Lambda_k(q, [A \to \alpha \bullet \beta])$ 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 $\Lambda_k(q, [A \to \alpha \bullet \beta])$ berechnet werden können :-)

Idee: Stelle ein Ungleichungssystem auf !!!

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

Beispiel:

$$
\begin{array}{lcl}
S & \rightarrow & A\,b\,B \mid B \\
A & \rightarrow & a \mid b\,B \\
B & \rightarrow & A
\end{array}
$$

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

$$
\begin{array}{lll}
q_0 = & \{[S' \rightarrow \bullet\, S], & q_2 = \delta(q_0, a) = \{[A \rightarrow a\bullet]\} \\
 & [S \rightarrow \bullet\, A\,b\,B], & \\
 & [A \rightarrow \bullet\, a], & q_3 = \delta(q_0, b) = \{[A \rightarrow b \bullet B], \\
 & [A \rightarrow \bullet\, b\,B], & [B \rightarrow \bullet\, A], \\
 & [S \rightarrow \bullet\, B], & [A \rightarrow \bullet\, a], \\
 & [B \rightarrow \bullet\, A]\} & [A \rightarrow \bullet\, b\,B]\}
\end{array}
$$

$$
q_1 = \delta(q_0, S) = \{[S' \rightarrow S\bullet]\} \qquad q_4 = \delta(q_0, B) = \{[S \rightarrow B\bullet]\}
$$

$$q_5 = \delta(q_0, A) = \{[S \to A \bullet b\, B], [B \to A \bullet]\}$$

$$q_6 = \delta(q_3, A) = \{[B \to A \bullet]\}$$

$$q_7 = \delta(q_3, B) = \{[A \to b\, B \bullet]\}$$

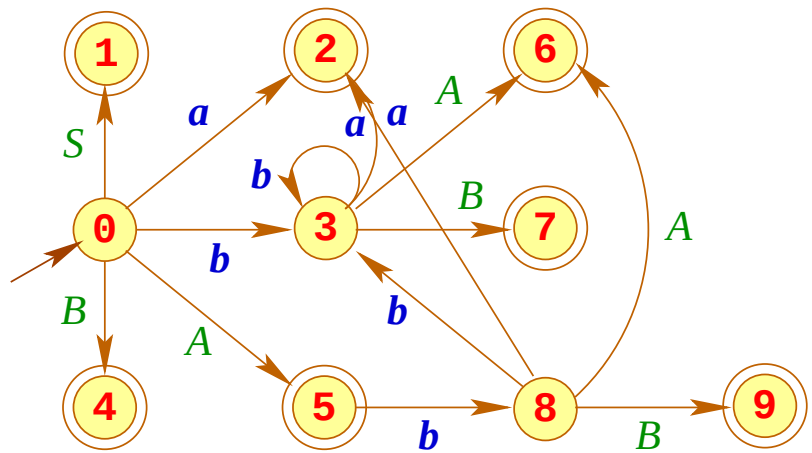
$$q_8 = \delta(q_5, b) = \{[S \to A\, b \bullet B], [B \to \bullet A], [A \to \bullet a], [A \to \bullet b\, B]\}$$

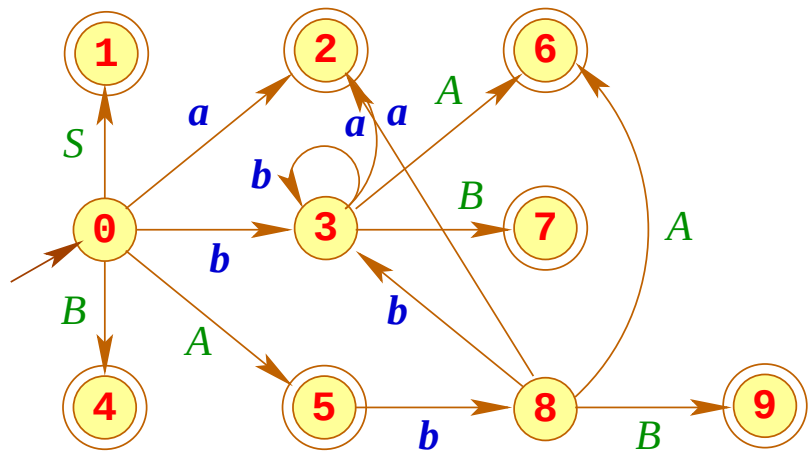
$$q_9 = \delta(q_8, B) = \{[S \to A\, b\, B \bullet]\}$$

Shift-Reduce-Konflikt:

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

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


1
2
6
0
3
7
4
5
8
9
S
a
a
a
A
b
B
A
b
b
B
A
b
B



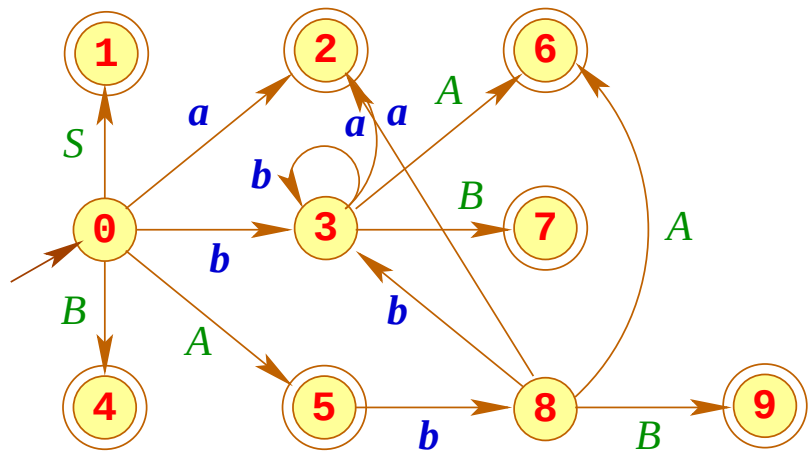
Ausschnitt des Ungleichungssystems:

$$\Lambda_1(q_5, [B \to A\bullet]) \supseteq \Lambda_1(q_0, [B \to \bullet A])$$

$$\Lambda_1(q_0, [B \to \bullet A]) \supseteq \Lambda_1(q_0, [S \to \bullet B])$$

$$\Lambda_1(q_0, [S \to \bullet B]) \supseteq \Lambda_1(q_0, [S' \to \bullet S])$$

$$\Lambda_1(q_0, [S' \to \bullet S]) \supseteq \{\epsilon\}$$



Ausschnitt des Ungleichungssystems:

$$\Lambda_1(q_1, [B \to A\bullet]) \supseteq \Lambda_1(q_0, [B \to \bullet A])$$

$$\Lambda_1(q_0, [B \to \bullet A]) \supseteq \Lambda_1(q_0, [S \to \bullet B])$$

$$\Lambda_1(q_0, [S \to \bullet B]) \supseteq \Lambda_1(q_0, [S' \to \bullet S])$$

$$\Lambda_1(q_0, [S' \to \bullet S]) \supseteq \{\epsilon\}$$

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

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: $A \Rightarrow L$ $B \Rightarrow R$ $a \Rightarrow \mathsf{id}$ $b \Rightarrow * \,/\, =$ liefert:

$$
\begin{array}{lcl}
S & \rightarrow & L = R \mid R \\
L & \rightarrow & \mathsf{id} \mid * R \\
R & \rightarrow & L
\end{array}
$$

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

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

Das verbesserte Ungleichungsssystem:

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

$\Longrightarrow$ ein reines Vereinigungsproblem :-))

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 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();
    }
}
```