

# 3. Kapitel

## Zerteilung



# Kapitel 3: Zerteilung

## 0. Einbettung

1. Theoretische Grundlage: Kontextfreie Grammatiken
  - Notation
  - Konkrete und abstrakte Syntax
  - Rekursiver Abstieg
  - Kellerautomaten
  - Elimination von Linksrekursion und  $\epsilon$ -Produktionen
  - Systematische Zerteilerkonstruktion: LL-, LR-Grammatiken
2. LL- und SLL-Grammatiken
3. LR-, SLR-Grammatiken
  - 3.1 LALR-Konstruktion
  - 3.2 Optimierungen und Komplexität
4. Fehlerbehandlung



## 3. Zerteilung

vorgegeben:

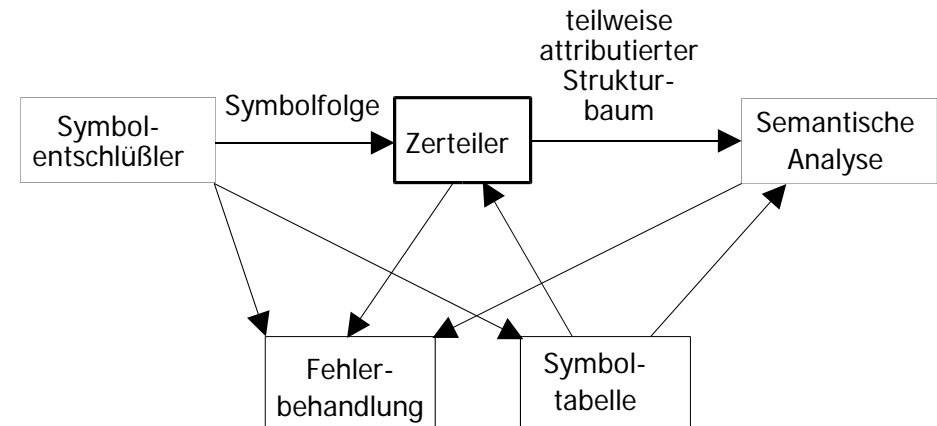
- Symbolfolge
- kontextfreie Grammatik (deterministisch?)

Aufgaben

- syntaktische Struktur bestimmen
- syntaktische Fehler melden, korrigieren (?)
- Ausgabe (**immer**): abstrakte Syntax (Rechts-/Linksableitung), Symbole (Bezeichner, Konstanten, usw.)



## 3. Einbettung des Zerteilers



### 3. ADT Zerteiler

gelieferte Operationen:

initialisiere  
beende  
zerteile

benötigte Operationen:

- Symbolentschlüssler/Symbolfolge  
nächstes\_Symbol();
- Fehlerbehandlung  
Fehlereintrag(Nr, pos)
- für Aufbau Strukturbaum  
produktion(nr), symbol(merkmal)



### 3. Aufgabe des Zerteilers, formal

**Gegeben:** Grammatik  $G=(\Sigma,N,P,Z)$  mit

$\Sigma$  Alphabet,  $N$  Nichtterminale,  $P$  Produktionen,  $Z$  Zielsymbol

**Gesucht:** Entscheidung, gehört Symbolfolge  $s$  zu  $L(G)$ ,

wenn ja, Produktionsfolge für Links-/Rechtsableitung  
wenn nein, Fehlerbehandlung zur Korrektur der Symbolfolge.

Unterscheide konkrete Syntax  $G_k$  und abstrakte Syntax  $G_a$ :

**Gesucht:** Entscheidung, gehört Symbolfolge  $s$  zu  $L(G_k)$ ,  
wenn ja, Produktionsfolge für Links-/Rechtsableitung für  $G_a$ ?

Beziehung zwischen  $G_k$  und  $G_a$ ?



### 3. Fragen

Wie wird Sprache erkannt?

Wie wird abstrakter Strukturbaum aufgebaut?

Was geschieht bei Fehlern?



### 3 Historie, kf Grammatiken + Verarbeitung

1955	Definition und Klassifikation (Chomsky und Bar Hillel)
1957-59	Kellerautomaten (Bauer&Samelson, sequentielle Formelübersetzung, 1959)
1961	formaler Zusammenhang kfGs-Kellerautomat (Öttinger)
1958-1966	kfGs und BNF setzen sich für die Syntax von Programmiersprachen durch (Algol 58, Algol 60, ...)
1960-1972	Verfahren des rekursiven Abstiegs (Glennie) und dessen theoretische Fundierung als LL-Grammatiken (auch heute noch oft neu erfunden!)
1963-1969	deterministische kfGs: beschränkte Operatorpräzedenz, LR, SLR, LALR,...
seit 1972	nichts wesentlich Neues außer Optimierung, Fehlerbehandlung



# Kapitel 3: Zerteilung

## 1. Theoretische Grundlage: Kontextfreie Grammatiken

- Notation
- Konkrete und abstrakte Syntax
- Rekursiver Abstieg
- Kellerautomaten
- Elimination von Linksrekursion und  $\epsilon$ -Produktionen
- Systematische Zerteilerkonstruktion: LL-, LR-Grammatiken
- 2. LL- und SLL-Grammatiken
- 3. LR-, SLR-Grammatiken
  - 3.1 LALR-Konstruktion
  - 3.2 Optimierungen und Komplexität
- 4. Fehlerbehandlung



# 3.1 Schreibweise der Produktionen

in der Theorie:  $A \rightarrow x / y / \dots, A \in N, x, y \in V^*, V = \Sigma \cup N$   
 praktisch: Backus-Naur-Form (BNF)

- Nichtterminale in spitzen Klammern,
- Terminale als Symbole oder wie Nichtterminale
- ::= statt  $\rightarrow$   
 $\langle \text{Ausdruck} \rangle ::= \langle \text{Term} \rangle \mid \langle \text{Ausdruck} \rangle + \langle \text{Term} \rangle$

Rechnereingabe: Erweiterte Backus-Naur-Form (EBNF)

- wie BNF, aber Bezeichner oft ohne spitze Klammern
- | (oder), . (Abschluß), ( ) (Gruppierung), [ ] (optional), \* (Wiederholung, auch 0-mal), + (Wiederholung, mindestens einmal) als Beschreibungssymbole
- Terminale durch Apostrophs o. ä. ausgezeichnet  
 $\text{Ausdruck} ::= \text{Term} ( '+' \text{Term} )^*$

Fortran-, Cobol-, Java-Beschreibung: Abarten von EBNF



## 3.1 EBNF mit Listennotation

Grammatik	::= Regel + .
Regel	::= Bezeichner ::= 'Ausdruck' .
Ausdruck	::= ( Element +    ' ' )   Ausdruck '    ' Atom .
Element	::= Einheit [ '*'   '+' ]   '[' Ausdruck ' ]'
Einheit	::= Atom   '(' Ausdruck ')' .
Atom	::= Bezeichner   Literal .

Ein Literal ist ein Text begrenzt durch Apostrophs.

$x // ' , ' '$  bedeutet ein oder mehrere  $x$ , getrennt durch Komma

Stärkste Bindung \*, +, dann ||, dann |, dann Gesamtregel



## 3.1 Beispielgrammatik

Ausdrücke

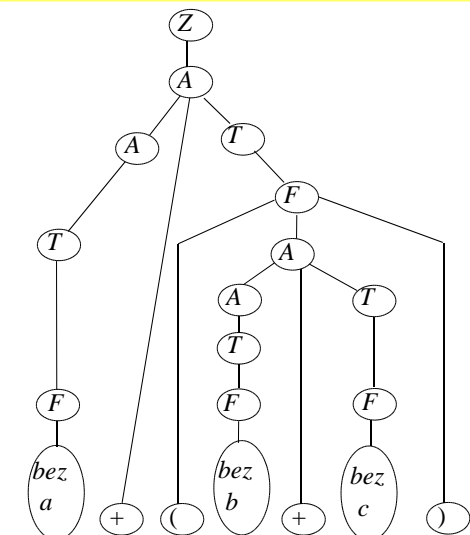
- (0)  $Z \rightarrow A$
- (1)  $A \rightarrow T \quad (2) A \rightarrow A + T$
- (3)  $T \rightarrow F \quad (4) T \rightarrow T * F$
- (5)  $F \rightarrow \text{bez} \quad (6) F \rightarrow ( A )$

EBNF:

- (0)  $Z \rightarrow A$ .
- (1)  $A \rightarrow T ( '+' T )^*$ .
- (2)  $T \rightarrow F ( '*' F )^*$ .
- (3)  $F \rightarrow \text{bez} / '( A )'$ .

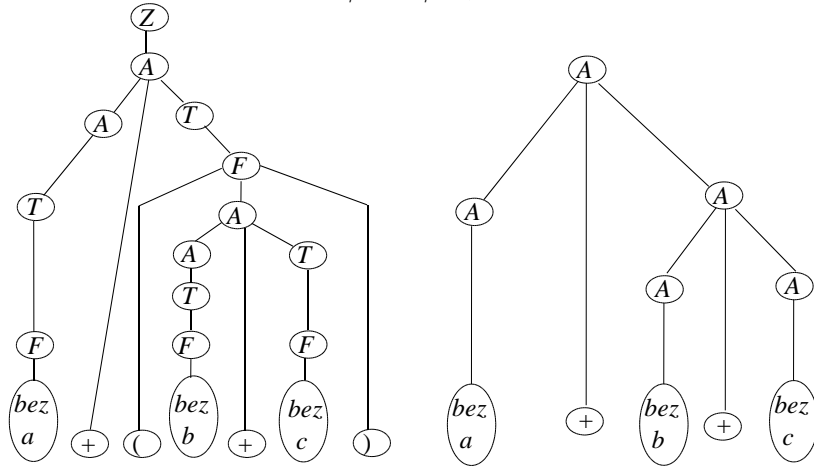
Ausdruck  $a + (b+c)$

konkreter Strukturbaum:



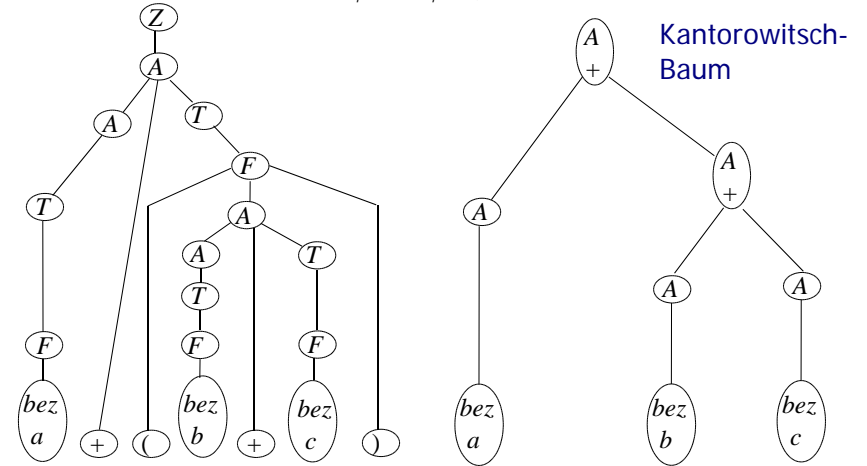
### 3.1 Konkrete und abstrakte Syntax

Prinzip der abstrakten Syntax: nur die für die Semantik wichtige Struktur behalten:  $A \rightarrow A + A \mid A * A \mid bez$



### 3.1 Konkrete und abstrakte Syntax

Prinzip der abstrakten Syntax: nur die für die Semantik wichtige Struktur behalten:  $A \rightarrow A + A \mid A * A \mid bez$



### 3.1 Abstrakte Syntax

Konkrete Syntax enthält redundante Information:

- Ketten- und Verteilerproduktionen  $A \rightarrow B$  bzw.  $A \rightarrow B \mid C \mid \dots$
- Wort- und Operatorsymbole, die spezifisch für die Produktion sind
- explizite Strukturinformation (Klammerung)

diese Ecken und Produktionen entfallen im Strukturbaum, außer sie tragen semantische Bedeutung

- z.B. beim Operandenzugriff: Unterschied Adresse/Wert

Produktionsnummer wird Eckentyp

Operatoren als Attribute der Ecke für den Ausdruck

### 3.1 Sonderfälle in abstrakter Syntax

Bezeichner:

- $A \rightarrow bez$  ist Kettenproduktion mit semantischer Bedeutung *leseInhalt* der Variablen *bez* (Unterschied zu  $bez := \dots$ )

Klammern in Fortran:

- Information eigentlich bereits in der Baumstruktur
- **aber** Klammern sind bindend (kein Umordnen erlaubt)
- sonst gilt eventuell Assoziativgesetz (umordnen erlaubt?)
- müssen als Operator gespeichert werden

Anweisungslisten in C:

- sind Verteilersymbole
- **aber** Strichpunkt-Operator legt Auswertungsfolge fest (auch ohne Datenabhängigkeiten), Code-Verschiebung verboten?

## 3.1 Abstrakte Syntax II

- abstrakte Syntax quellsprachenunabhängig?
  - Programmstruktur in semantischer Analyse aufgearbeitet, danach nur noch Prozeduren interessant
  - Prozeduraufrufe nur bezüglich Parameterübergabe unterschiedlich
  - Ablaufsteuerung identisch, eventuelle Ausnahme: Zählschleifen
  - Ausnahmebehandlung in allen modernen Sprachen identisch
  - Zuweisung, Ausdrucksoperatoren, usw.: identisch, manchmal vielleicht Ergänzungen erforderlich
- Konsequenz: weitere Verarbeitung (Transformation, Optimierung, Codegen.) weitgehend unabhängig von der Quellsprache
  - Systeme: UNCOL, ANDF, Dotnet
  - Dotnet kann als Postfixcodierung von UNCOL angesehen werden



## 3.1 Anknüpfungen

Ausgabe des Zerteilers: Produktionen von  $G_a$  und Symbole

Methode: Anknüpfung der Ausgaberroutinen in die Grammatik eintragen

**Syntaktische Anknüpfungen:** %Ausgabe

**Symbol Anknüpfungen:** &Ausgabe

Beachte: Symbole werden in der Reihenfolge abgenommen, in der sie in der Symbolfolge erscheinen



## 3.1 Ausgabe von Postfix- oder Präfixform

Ausgaberroutinen:

*addop:* gib aus '+'  
*mulop:* gib aus '\*'  
*bezeichner:* gib aus *bez*  
*merke:* merke *bez*  
*bez\_aus:* gib gemerkten *bez* aus

Postfixform, d.h. abstrakter Syntaxbaum als Rechtsableitung:

- (1)  $A \rightarrow T ( '+' T \%addop )^*$ .
- (2)  $T \rightarrow F ( '*' F \%mulop )^*$ .
- (3)  $F \rightarrow bez \&bezeichner | '( A ' )'$ .

Präfixform, d.h. abstrakter Syntaxbaum als Linksableitung:

- (1)  $A \rightarrow T ( '+' \%addop \%bez\_aus T )^* \%bez\_aus$ .
- (2)  $T \rightarrow F ( '*' \%mulop \%bez\_aus F )^* \%bez\_aus$ .
- (3)  $F \rightarrow bez \&merke | '( A ' )'$ .



## 3.1 Rekursiver Abstieg

Einfachstes und intuitives Verfahren:

Verstehe die Produktionen als rekursives Programm mit Rücksetzmöglichkeit bei Fehlentscheidungen

Konstruiere Linksableitung

Füge explizit an entsprechenden Stellen ein:

- Anknüpfungen
- (Setzen von Attributen)
- notwendige Fehlermeldungen



## 3.1 Zerteiler für Beispielgrammatik I

Anwendung von rekursivem Abstieg auf die Beispielgrammatik

- (0)  $Z \rightarrow A$ .
- (1)  $A \rightarrow T ('+T)^*$ .
- (2)  $T \rightarrow F ('*F)^*$ .
- (3)  $F \rightarrow \text{bez}/('A')$ .

mit Aufbau des Strukturbaums als Linksableitung

Ecken des Strukturbaums gehören zur Klasse

```
class AST {
    AST links, rechts; // die beiden Söhne
    OPER operator; // kennzeichnet die Produktion
    SYMBOL merkmal; // Merkmal von Bezeichnern
    AST(SYMBOL s) { this.merkmal = s; };
    AST(OPER op) { this.operator = op; };
}; // class AST
```

i.a. ist AST eine abstrakte Klasse, von der die verschiedenen Eckentypen (hier: operator- und symbol-Ecke) erben



## 3.1 Zerteiler für Beispielgrammatik II

```
class Syntaxanalyse {
    zerteile(): AST {s=nextSymbol(); return Z();}
    Z(): AST {return A();}
    A(): AST {
        AST res = T(); // merke 1. Operand
        while ( s == "+" ) {s=nextSymbol();
            AST res1 = new AST(plus);
            res1.left = res; res1.right = T(); res = res1;}
        return res; }
    T(): /* analog A */
    F(): AST{
        if (class(s)== bez){AST res = new AST(s);s=nextSymbol(); return res;}
        if (class(s)==KlammerAuf){s=nextSymbol(); AST res = A();
            if (class(s)==KlammerZu) s=nextSymbol();
            else Fehlerbehandlung.Fehlereintrag(KlammerZuFehlt,s.pos);}
        else Fehlerbehandlung.Fehlereintrag(unzulässigesSymbol,s.pos);
        return res;}
    }
}
```



## 3.1 Annahmen für die Zerteilung

Syntax ist kontextfrei

- eigentlich ist sie kontext-sensitiv
- aber kontext-sensitive Grammatiken nicht in linearer Zeit zerteilbar (Kontextfreiheit ist selbsterfüllende Prophezeiung)
- der über die kontextfreie Grammatik hinausgehende Teil der Syntax heißt im Übersetzerbau **statische Semantik**

Syntax ist deterministisch kontextfrei

- keine wesentliche Einschränkung, da auch vom menschlichen Leser erwünscht

keine Rückkopplung zur Symbolentschlüsselung

- sonst gäbe es mehrere Grundzustände des Symbolentschlüßlers, gesteuert vom Zerteiler

keine Rückkopplung semantische Analyse - Zerteilung

- **typunabhängige Zerteilung**: Zustände des Zerteilers unabhängig von der Namens- und Typanalyse



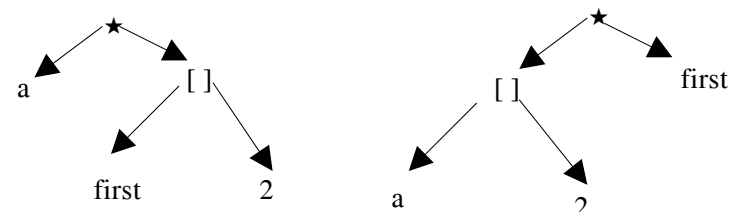
## 3.1 Typunabhängiges Zerteilen

Typunabhängiges Zerteilen

- Zerteilen ohne Kenntnis über Typen von Symbolen
- ist üblich, aber nicht immer ausreichend

Typabhängiges Zerteilen

- Bsp: ADA  $a * \text{first}(2)$



### 3.1 Typabhängiges Zerteilen

Beispiel: Formate in FORTRAN

- `print(r 20, real_const)`
- `r 20` ist Format und muß anders behandelt werden, sonst `r` Bezeichner und `20` ganze Zahl

Zerteiler umschaltbar, um Formate zu bearbeiten

- D.h., es gibt zwei verschiedene Zerteiler
- Erst semantische Analyse erkennt Bezeichner `print`

Umschaltung also semantik- (oder typ-) gesteuert

Ähnliche Probleme in ABAP/4



### 3.1 Kellerautomaten

$A = (\Sigma, Q, R, q_0, F, S, s_0)$

$\Sigma$  Eingabealphabet (Symbole)

$Q$  Zustandsmenge

$R$  Menge von Regeln  $sqx \rightarrow s'q'x', s, s' \in S^*, q, q' \in Q, x \in \Sigma^*, x = x''x'$

- $q_0$  Anfangszustand,  $F \subseteq Q$  Menge von Endzuständen
- $S$  Kelleralphabet,  $s_0 \in S$  Anfangszeichen im Keller

- **Konfiguration:**  $\underline{s}q\underline{x}$ ,  $\underline{s}$  vollständiger Kellerinhalt,  $\underline{x}$  restliche Eingabe

- **Anfangskonfiguration:**  $s_0qy$ ,  $y$  vollständige Eingabe

- **Regel**  $sqx \rightarrow s'q'x'$  **anwendbar**, wenn  $\underline{s} = \underline{s's}$ ,  $\underline{x} = \underline{xx'}$

$\underline{s'}$	$s$	$q$	$x$	$\underline{x'}$
$\underline{s}$	$s'$	$q'$	$x'$	$\underline{x}$

- Folgekonfiguration:  $\underline{s's'q'x'x'}$

- **Halt** bei Konfiguration  $sq$ ,  $q \in F$ , Eingabe vollständig gelesen  
praktisch **Endezeichen #** erforderlich, Halt bei  $sq\#$



### 3.1 Kontextfreie Grammatik und Kellerautomaten

**Satz:** Für jede kontextfreie Grammatik  $G$  gibt es einen (nicht-deterministischen) Kellerautomaten  $A$  mit  $L(A)=L(G)$

⇒ das Akzeptionsproblem für kontextfreie Sprachen ist entscheidbar.

**Aber:** Sprachinklusion und Gleichheit nicht entscheidbar

⇒ keine eindeutige Normalform

**Aber:** Aufwand i.a.  $O(n^3)$

⇒ praktisch nur Teilklassen mit linearem Aufwand brauchbar, dazu Grammatik-Umformungen erforderlich



### 3.1 Textmengen

$$k : x = x\#, \quad \text{falls} \quad |x| < k$$

$$k : x = x_1 \dots x_k, \quad \text{falls} \quad x = x_1 \dots x_m \wedge m \geq k$$

$$\text{Anf}_k(x) = \{u / \exists y \in \Sigma^* \text{ so daß } x \Rightarrow^* y, u = k : y\}$$

$$\text{Anf}'_k(x) = \{u / u \in \text{Anf}_k(x) \wedge \neg \exists A \in N, y \in \Sigma^* \text{ so daß } x \Rightarrow^R Auy \Rightarrow uy\}$$

$$\text{Folge}_k(x) = \{u / \exists y \in V^* \text{ so daß } Z \Rightarrow^* mxy, u \in \text{Anf}_k(y)\}$$



## 3.1 Beseitigung von $\epsilon$ -Produktionen

**Satz:** Für jede kfG  $G$  mit  $\epsilon$  Produktionen gibt es eine kfG  $G'$  ohne  $\epsilon$  Produktionen mit  $L(G) - \{\epsilon\} = L(G')$  und umgekehrt.

Technik dazu: Linksfaktorisierung



## 3.1 Linksfaktorisierung

Produktionen  $X \rightarrow Yb / Yc$  mit gleicher LS und gemeinsamem Anfang  $Y$  kann man nicht mit rekursivem Abstieg verarbeiten, wenn Länge  $|y|, Y \Rightarrow^* y$ , unbeschränkt,  $|y| \geq 0$ .

**Lösung:** den gemeinsamen Anfang ausklammern  
ersetze  $X \rightarrow Yb / Yc$  durch  $X \rightarrow YX', X' \rightarrow b / c$

Analog kann man rechtsfaktorieren (seltener benötigt).



## 3.1 Systematische Zerteilerkonstruktion

- Es gibt weit mehr als 25 verschiedene Techniken zur Zerteilerkonstruktion, vgl. Aho&Ullman, *The Theory of Parsing and Compiling*, 2 Bde, 1972
- Nur zwei Techniken, LL und LR, haben die Eigenschaften:
  - Der Zerteiler liest die Quelle **einmal** von links nach rechts;
  - Der Zerteiler erkennt einen Fehler beim ersten Zeichen  $t$ , das nicht zu einem Satz der Sprache gehören kann.  $t$  heißt **zerteilerdefinierte Fehlerstelle** (*parser defined error*):  
Wenn  $x \notin L(G)$  und der Zerteiler erkennt den Fehler beim Zeichen  $t$ ,  $x = x'tx''$ , so gibt es einen Satz  $y \in L(G)$  mit  $y = x'y'$ .
- Alternative: Erkennen des Fehlers einige Zeichen später, keine syntaktische Fehlerlokalisierung möglich.



## 3.1 Herleitung der LL- und LR-Zerteiler

- gegeben Grammatik  $G=(\Sigma, N, P, Z)$ ,  $V = \Sigma \cup N$ , konstruiere **indeterministischen** Kellerautomat mit genau einem Zustand  $q$ , angesetzt auf Eingabe  $x$

Für LL: (prädiktiv)

$tqt \rightarrow q, t \in T$

$Xq \rightarrow x_n \dots x_1 q, X \rightarrow x_1 \dots x_n \in P$

Für LR: (reduzierend)

$qt \rightarrow tq, t \in T$

$x_1 \dots x_n q \rightarrow Xq, X \rightarrow x_1 \dots x_n \in P$

- mache Kellerautomat **deterministisch** durch Hinzunahme Rechtskontext, also Vorhersage  $Xqx' \rightarrow x_n \dots x_1 qx'$  bzw. Reduktion  $x_1 \dots x_n qx' \rightarrow Xqx'$ ,  $x'$  Anfang des unverarbeiteten Eingaberests
- **deterministisch machen geht nur für eingeschränkte Grammatikklassen**





## 3.1 Nichtdeterministische LL- und LR-Zerteiler

Für LL: (prädiktiv)

Vergleich (compare):

$$tqt \rightarrow q, t \in T$$

Vorhersage (produce):

$$Xq \rightarrow x_n \dots x_1 q, X \rightarrow x_1 \dots x_n \in P$$

Für LR: (reduzierend)

Schift (shift):

$$qt \rightarrow tq, t \in T$$

Reduktion (reduce):

$$x_1 \dots x_n q \rightarrow Xq, X \rightarrow x_1 \dots x_n \in P$$

top-down Zerteiler

bottom-up Zerteiler

vom Startsymbol zum Wort

vom Wort zum Startsymbol

Anmerkung: Der Zustand ist bedeutungslos.



## Kapitel 3: Zerteilung

### 1. Theoretische Grundlage: Kontextfreie Grammatiken

Notation

Konkrete und abstrakte Syntax

Rekursiver Abstieg

Kellerautomaten

Elimination von Linksrekursion und  $\epsilon$ -Produktionen

Systematische Zerteilerkonstruktion: LL-, LR-Grammatiken

### 2. LL- und SLL-Grammatiken

### 3. LR-, SLR-Grammatiken

3.1 LALR-Konstruktion

3.2 Optimierungen und Komplexität

### 4. Fehlerbehandlung



## 3.2 $LL(k)$ -Grammatiken

Für  $k \geq 1$  heißt eine kfG  $G=(\Sigma, N, P, Z)$  eine  $LL(k)$ -Grammatik, wenn

für alle Paare von Ableitungen

$$Z \Rightarrow_L \mu A \chi \Rightarrow \mu v \chi \Rightarrow^* \mu \gamma \quad \mu, \gamma \in T^* \quad v, \chi \in V^*, A \in N$$

$$Z \Rightarrow_L \mu A \chi' \Rightarrow \mu \omega \chi' \Rightarrow^* \mu \gamma' \quad \mu, \gamma' \in T^* \quad \omega, \chi', \in V^*$$

gilt:

$$(k : \gamma = k : \gamma') \Rightarrow v = \omega.$$

Also: Aus den nächsten  $k$  Zeichen kann unter Berücksichtigung des Kellerinhalts die nächste anzuwendende Produktion eindeutig vorhergesagt werden.

Die  $k$  Zeichen können aus der Produktion resultieren oder ganz oder teilweise dem Folgetext angehören, z.B. bei  $\epsilon$ -Produktionen.



## 3.2 Beispiele von $LL$ -Grammatiken

$$A \rightarrow TA', A' \rightarrow \epsilon / +TA', T \rightarrow FT', T' \rightarrow \epsilon / *FT', F \rightarrow bez / ('A') \diamond$$

ist  $LL(1)$ .

$Z \rightarrow aAab / bAbb, A \rightarrow \epsilon / a$  ist  $LL(2)$ , nicht  $LL(1)$ : Vorschau  $aa, ab, bb$  entscheidet unter Berücksichtigung der Produktion für  $Z$  über die Produktion für  $A$ .

$$Z \rightarrow X, X \rightarrow Y / bYa, Y \rightarrow c / ca \text{ ist } LL(3).$$

$Z \rightarrow X, X \rightarrow Yc / Yd, Y \rightarrow a / bY$  ist für kein  $k$   $LL(k)$ ; aber Linksfaktorisieren macht daraus  $LL(1)$ .

Anweisungen, die mit Schlüsselwort `while`, `if`, `case`, usw. beginnen, sind mit  $LL(1)$ -Technik vorhersehbar. Bei Beginn mit Bezeichner sind Umformungen nötig.



## 3.2 Satz über Linksrekursion

**Satz:** Eine linksrekursive kfG ist für kein  $k \in \mathbb{N}$   $LL(k)$ .

**Beweisidee:**

Seien  $A \rightarrow Ax$  und  $A \rightarrow y$  linksrekursive bzw. terminierende Regeln.

Jeder  $k$ -Anfang der terminierenden Regel ist auch  $k$ -Anfang der linksrekursiven Regel.



## 3.2 Elimination von Linksrekursion

**Satz:** Für jede kfG  $G$  mit linksrekursiven Produktionen gibt es eine kfG  $G'$  ohne Linksrekursion mit  $L(G)=L(G')$ .

• **Konstruktion:**

• Numeriere Nichtterminale  $X_1, \dots, X_n$ . Ziel:  $i < j$ , wenn  $X_i \rightarrow X_j x \in P$

• Für  $i=1, \dots, n$

• Für  $j=1, \dots, n-1$  ersetze  $X_i \rightarrow X_j x$  durch  $\{X_i \rightarrow y_j x / X_j \rightarrow y_j \in P\}$

(danach  $i \leq j$ , wenn  $X_i \rightarrow X_j x \in P$ )

• ersetze die Produktionsmengen  $\{X_i \rightarrow X_i x\} \cup \{X_i \rightarrow z / z \neq X_i z'\}$  durch  $\{Y_i \rightarrow x Y_i / X_i \rightarrow X_i x \in P\} \cup \{Y_i \rightarrow \varepsilon\} \cup \{X_i \rightarrow z Y_j / X_i \rightarrow z \in P \text{ und } z \neq X_i z'\}$  mit einem neuen Nichtterminal  $Y_i$ . (Numerierung der  $Y_i$  mit  $n+1, n+2, \dots$ )

• **Beachte:** in Schritt 2 Ersetzung durch

$\{Y_i \rightarrow x, Y_i \rightarrow x Y_i / X_i \rightarrow X_i x \in P\} \cup \{X_i \rightarrow z Y_j / X_i \rightarrow z \in P \text{ und } z \neq X_i z'\}$

ohne  $\varepsilon$ -Produktionen möglich, wenn  $x$  nicht mit  $X_j, j \leq i$ , beginnt.



## 3.2 Beispiel

•  $A \rightarrow T / A + T, T \rightarrow F / T * F, F \rightarrow \text{bez} / (A)$

ist linksrekursiv

• **Ersetzung:** Schritt 1 leer, Schritt 2:  $A \rightarrow T / A + T$  durch

$A \rightarrow T A', A' \rightarrow \varepsilon / + T A'$  ersetzen;  $T \rightarrow F / T * F$  analog.

Dies entspricht der EBNF

$A \rightarrow T ('+' T)^*, T \rightarrow F ('*' F)^*, F \rightarrow \text{bez} / '(' A ')'$ .

• **Andere mögliche Ersetzung**  $A \rightarrow T / T A', A' \rightarrow + T / T A'$

• **Vorsicht!**

Die Ersetzung durch  $A \rightarrow T / T + A$  ist **semantisch unzulässig!**

• Sie transformiert Links- in Rechtsassoziativität, verändert also die semantisch bedeutungsvolle Struktur.

• Beseitigung von Linksrekursion bei rekursivem Abstieg nötig für alle

Anweisungen, die mit  $\langle \text{Bezeichner} \rangle$   $\langle \text{Operator} \rangle$  anfangen können (Zuweisungen, Ausdrücke)



## 3.2 $SLL(k)$ -Grammatiken

Für  $k \geq 1$  heißt eine kf Grammatik  $G=(\Sigma, N, P, Z)$  eine  $SLL(k)$ -Grammatik (**starke LL-Grammatik**), wenn für alle Paare von Ableitungen

$$\begin{aligned} Z \Rightarrow^L \mu A \chi \Rightarrow \mu \nu \chi &\Rightarrow^* \mu \gamma & \mu, \gamma \in T^* & \nu, \chi \in V^*, A \in N \\ Z \Rightarrow^L \mu' A \chi' \Rightarrow \mu' \omega \chi' &\Rightarrow^* \mu' \gamma' & \mu', \gamma' \in T^* & \omega, \chi' \in V^* \end{aligned}$$

gilt:

$$(k : \gamma = k : \gamma') \Rightarrow \nu = \omega.$$

Also: Aus den nächsten  $k$  Zeichen kann **ohne** Berücksichtigung des Kellerinhalts die nächste anzuwendende Produktion eindeutig vorhergesagt werden.



## 3.2 SLL-Bedingung

**Satz:** Eine Grammatik ist genau dann eine SLL(k)-Grammatik, wenn für alle Paare von Produktionen  $A \rightarrow x / x'$ ,  $x \neq x'$ , die SLL(k)-Bedingung gilt:

$$\text{Anf}_k(x \text{ Folge}_k(A)) \cap \text{Anf}_k(x' \text{ Folge}_k(A)) = \emptyset$$

**Beweis:** trivial

- Also: SLL(k)-Eigenschaft durch Berechnung von  $\text{Anf}_k$ - und  $\text{Folge}_k$ -Mengen einfach nachzuprüfen.
- Wenn aus  $x$ ,  $x'$  nur terminale Zeichenreihen mit mindestens  $k$  Zeichen ableitbar sind, trägt  $\text{Folge}_k(A)$  nichts zum Ergebnis bei und kann entfallen.
- Anwendung: Für  $k = 1$  entfällt  $\text{Folge}_1(A)$  bei  $\varepsilon$ -freien Grammatiken.



## 3.2 LL(1) und SLL(1)

**Satz:** Jede SLL(k)-Grammatik ist auch eine LL(k)-Grammatik.

**Satz:** Jede LL(1)-Grammatik ist eine SLL(1)-Grammatik.

**Beweis:** gegeben  $A \rightarrow x / x'$ ,  $x \neq x'$ , und ein Zeichen  $t \in \text{Anf}_1(x \text{ Folge}_1(A)) \cap \text{Anf}_1(x' \text{ Folge}_1(A))$ .

Fallunterscheidung

- $t \in \text{Anf}_1(x)$ ,  $t \in \text{Anf}_1(x')$
- $\varepsilon \in \text{Anf}_1(x)$ ,  $t \in \text{Anf}_1(x')$ ,  $t \in \text{Folge}_1(A)$
- $t \in \text{Anf}_1(x)$ ,  $\varepsilon \in \text{Anf}_1(x')$ ,  $t \in \text{Folge}_1(A)$
- $\varepsilon \in \text{Anf}_1(x)$ ,  $\varepsilon \in \text{Anf}_1(x')$ ,  $t \in \text{Folge}_1(A)$

Alle 4 Fälle nach LL(1)-Definition unzulässig. Vierter Fall sogar mehrdeutig.

**Satz nicht auf  $k > 1$  verallgemeinerbar:**

$Z \rightarrow aAab / bAbb$ ,  $A \rightarrow \varepsilon / a$  ist LL(2), aber nicht SLL(2).



## 3.2 Situationen

**Ziel:** bei Prüfung der Anwendbarkeit von Regeln  $sqx \rightarrow s'q'x'$

$sq$  in einem Zustandssymbol codieren (Prüfung mehrerer Symbole im Keller vermeiden)

**Lösungsidee:**

- bei LL und LR ist  $s$  rechte bzw. linke Seite einer Produktion  $X \rightarrow x_1 \dots x_n$
- Übergänge  $tqt \rightarrow q$  bzw.  $qt \rightarrow qt$  sind nur zulässig, wenn in der Produktion ein Terminalzeichen  $t$  ansteht,  $x_1 \dots x_n = x'x''$  und  $x'' = t\underline{x''}$
- also: ersetze  $sqx$  durch **Situation**  $[X \rightarrow x' \approx x''; x]$ , die durch den Punkt anzeigt, wie weit die Produktion abgearbeitet ist.
- Situationen  $[X \rightarrow \approx x''; x]$  oder  $[X \rightarrow x' \approx; x]$  sind erlaubt und notwendig.
- Verwende Situationen als Zustände und als Kellersymbole.
- Situationen heißen englisch *items*.



## 3.2 Formale Konstruktion LL(k)-Automat

1. **Initial**  $Q = \{q_0\}$  und  $R = \emptyset$ , mit  $q_0 = [Z \rightarrow \bullet S; \#]$ .

Anfangszustand und erster Kellerezustand  $q_0$ . **Hinweis:**  $\text{Folge}_k(Z) = \{\#\}$ .

2. Sei  $q = [X \rightarrow \mu \bullet v; \Omega] \in Q$  und noch nicht betrachtet.

3. **Wenn**  $v = \varepsilon$  setze  $R := R \cup \{q \varepsilon \rightarrow \varepsilon\}$

Auskellern  $q'q \rightarrow q'$  mit beliebigem  $q' \approx$

4. **Wenn**  $v = t\gamma$  mit  $t \in T$  und  $\gamma \in V^*$ , setze  $q^* = [X \rightarrow \mu t \bullet \gamma; \Omega]$ .

Setze  $Q := Q \cup \{q^*\}$  und  $R := R \cup \{qt \rightarrow q^*\}$ .

5. **Wenn**  $v = B\gamma$  mit  $B \in N$  und  $\gamma \in V^*$ , setze  $q^* = [X \rightarrow \mu B \bullet \gamma; \Omega]$  und

$H = \{[B \rightarrow \bullet \beta_i; \text{Anf}_k(\gamma \Omega)] / B \rightarrow \beta_i \in P\}$ .

**Hinweis:**  $1 \leq i \leq m$ , wenn es  $m$  Produktionen mit linker Seite  $B$  gibt.

Setze  $Q := Q \cup \{q^*\} \cup H$  und  $R := R \cup \{q\tau_i \rightarrow q^*h_i\tau_i / h_i \in H, \tau_i \in \text{Anf}_k(\beta_i \gamma \Omega)\}$ .

6. **Wenn alle**  $q \in Q$  betrachtet wurden, **stop**. **Sonst**, gehe zu (2).



## 3.2 Formale Konstruktion $SLL(k)$ -Automat

- Initial**  $Q = \{q_0\}$  und  $R = \emptyset$ , mit  $q_0 = [Z \rightarrow \bullet S; \#]$ .  
Anfangszustand und erster Kellerzustand  $q_0$ . **Hinweis:**  $Folge_k(Z) = \{\#\}$ .
- Sei  $q = [X \rightarrow \mu \bullet v; \Omega] \in Q$  und noch nicht betrachtet.
- Wenn**  $v = \varepsilon$  setze  $R := R \cup \{q \varepsilon \rightarrow \varepsilon\}$   
Auskellern  $q' \alpha \rightarrow q' \ominus$  mit beliebigem  $q' \approx$
- Wenn**  $v = t\gamma$  mit  $t \in T$  und  $\gamma \in V^*$ , setze  $q^* = [X \rightarrow \mu t \bullet \gamma; \Omega]$ .  
Setze  $Q := Q \cup \{q^*\}$  und  $R := R \cup \{qt \rightarrow q'\}$ .
- 5'.** **Wenn**  $v = B\gamma$  mit  $B \in N$  und  $\gamma \in V^*$ , setze  $q^* = [X \rightarrow \mu B \bullet \gamma; \Omega]$  und  
 $H = \{[B \rightarrow \bullet \beta_i; Folge_k(B)] / B \rightarrow \beta_i \in P\}$ .  
**Hinweis:**  $1 \leq i \leq m$ , wenn es  $m$  Produktionen mit linker Seite  $B$  gibt.  
Setze  $Q := Q \cup \{q^*\} \cup H$  und  $R := R \cup \{q\tau_i \rightarrow q' h_i \tau_i / h_i \in H, \tau_i \in Anf_k(\beta_i Folge_k(B))\}$ .
- 6.** **Wenn alle**  $q \in Q$  betrachtet wurden, **stop**. **Sonst**, gehe zu (2).



## 3.2 Formale Konstruktion $SLL(k)$ -Automat

Einzig Regel 5 der  $LL(k)$  Konstruktion ändert sich:

- 5'.** **Wenn**  $v = B\gamma$  mit  $B \in N$  und  $\gamma \in V^*$ , setze  $q^* = [X \rightarrow \mu B \bullet \gamma; \Omega]$  und  
 $H = \{[B \rightarrow \bullet \beta_i; Folge_k(B)] / B \rightarrow \beta_i \in P\}$ .  
**Hinweis:**  $1 \leq i \leq m$ , wenn es  $m$  Produktionen mit linker Seite  $B$  gibt.  
Setze  $Q := Q \cup \{q^*\} \cup H$  und  $R := R \cup \{q\tau_i \rightarrow q' h_i \tau_i / h_i \in H, \tau_i \in Anf_k(\beta_i Folge_k(B))\}$ .



## 3.2 $LL(1)$ -Zerteiler mit rekursivem Abstieg

- Definiere Prozedur  $X$  für alle Nichtterminale  $X$
- Für alternative Produktionen  $X \rightarrow X_1, \dots, X \rightarrow X_n$  sei Rumpf von  $X$ 

```
switch s {
  case s ∈ Anf(X1 Folge(X)) : Code für X1 ;
  ...
  case s ∈ Anf(Xn Folge(X)) : Code für Xn ;
  default : Fehler(...); }
```
- Für rechte Seite  $X_i = Y_1 \dots Y_m$  erzeuge:
 

```
C1; ... ; Cm; return
```

 Es gilt  $C_i =$ 
  - if (s ==  $Y_i$ ) s = nächstesSymbol() else Fehler(...);  
wenn  $Y_i \in T$
  - $Y_i()$ ; wenn  $Y_i \in N$



## 3.2 Zerteiler aus Grammatik in EBNF

Nichtterminal	$X$	$X()$ ;
Terminal	$t$	if (symbol == t) {nächstesSymbol();} else Fehler();
Option	$[X]$	if (symbol ∈ Anf(X)) X();
Iteration	$X^*$	repeat X(); until !(symbol ∈ Anf(X));
	$X^+$	while (symbol ∈ Anf(X)) X();
Liste	$X   d$	$X()$ ; while (symbol ∈ Anf(d)) { d; X(); }
Anknüpfung	$t \& Y$	if (symbol == t) {Y(); nächstesSymbol();} else Fehler();
	$\%Z$	Z();



## 3.2 Zerteiler für Beispielgrammatik

```
class Syntaxanalyse {
zerteile(): AST {s=nextSymbol(); return Z();}
Z(): AST {return A();}
A(): AST {
  AST res = T(); // merke 1. Operand
  while ( s == "+" ) {s=nextSymbol();
    AST res1 = new AST(plus);
    res1.left = res; res1.right = T(); res = res1; }
  return res; }
T(): /* analog A */
F(): AST{
  if (class(s)== bez){AST res = new AST(s);s=nextSymbol(); return res;}
  if (class(s)==KlammerAuf){
    s=nextSymbol(); AST res = A();
    if (class(s)==KlammerZu) s=nextSymbol();
    else Fehlerbehandlung.Fehlereintrag(KlammerZuFehlt,s.pos); }
  else Fehlerbehandlung.Fehlereintrag(unzulässigesSymbol,s.pos);
  return res;}
}
```



## 3.2 Praxis des rekursiven Abstiegs

- Einfügung von **Anknüpfungen**: Anknüpfung formal wie Produktion  $A \rightarrow \varepsilon$  behandeln, statt der Prozedur für ein Nichtterminal  $A$  die Ausgabeprozedur aufrufen.
- Rekursiver Abstieg baut Linksableitung auf. Vorteil: beim Aufbau bereits erste **Berechnungen von semantischen Attributen möglich** (s. Kapitel "Semantische Analyse").
- **Problem**: Durch die Handprogrammierung können leicht während der Wartung syntaktische Eigenschaften eingeschleust werden, die die **Systematik der Syntax und die Unabhängigkeit Syntax-Semantik zerstören**. Negativbeispiel: ABAP 4
- Rekursiver Abstieg kann auch **tabellengesteuert** implementiert werden! Zerteiler wird Interpretierer der Tabelle. Vorteile: Vermeidung von Prozeduraufrufen, einfachere Fehlerbehandlung. Nachteil: nicht von Hand programmierbar.



## Kapitel 3: Zerteilung

### 1. Theoretische Grundlage: Kontextfreie Grammatiken

Notation

Konkrete und abstrakte Syntax

Rekursiver Abstieg

Kellerautomaten

Elimination von Linksrekursion und  $\varepsilon$ -Produktionen

Systematische Zerteilerkonstruktion: LL-, LR-Grammatiken

### 2. LL- und SLL-Grammatiken

### 3. LR-, SLR-Grammatiken

3.1 LALR-Konstruktion

3.2 Optimierungen und Komplexität

### 4. Fehlerbehandlung



## 3.3 Motivation für LR/LALR

- LR basierte Zerteiler sind die mächtigsten, die noch effizient berechenbar sind und es gibt ausgreife Generatoren.

- Für den yacc-Generator produziert die Grammatik

```
%token IF ELSE THEN IDENT
```

```
%%
```

```
Statement: IfStatement | Expr ;
```

```
Expr: IDENT;
```

```
IfStatement: IF Expr THEN Statement |
```

```
IF Expr THEN Statement ELSE Statement ;
```

- **allerdings folgende Fehlerdiagnose**

```
state 7 contains 1 shift/reduce conflict.
```

```
...
```

```
state 7
```

```
IfStatement -> IF Expr THEN Statement . (rule 4)
```

```
IfStatement -> IF Expr THEN Statement . ELSE Statement (rule 5)
```

```
ELSE shift, and go to state 8
```

```
ELSE [reduce using rule 4 (IfStatement)]
```

```
$default reduce using rule 4 (IfStatement)
```



### 3.3 Motivation für LR/LALR

- **Frage:** Wie kann man diese Fehlerdiagnose verstehen?
  - Zweck aller nachfolgenden Ausführungen ist nur darauf zu beziehen.
  - Detailliertes Wissen über automatische Konstruktion vernachlässigbar
  - **Aber:** Intuitive „Hand“konstruktion muss beherrscht werden.
- **Lösungen:**
  - Grammatik ändern: abschließendes „end“ einführen (meist unzulässig)
  - Ausfaktorisieren:
 

```
%token IF ELSE THEN IDENT
%%
Statement: IfStatement | Expr ;

Expr:      IDENT;

IfStatement:
    IF Expr THEN Statement |
    IF Expr THEN IfThenElseStat ELSE Statement ;

IfThenElseStat:
    IF Expr THEN IfThenElseStat ELSE IfThenElseStat |
    Expr ;
```

### 3.1 Wdh. Nichtdeterministische LL- und LR-Zerteiler

<b>Für LL: (prädiktiv)</b>	<b>Für LR: (reduzierend)</b>
Vergleich (compare): $tqt \rightarrow q, t \in T$	Schift (shift): $qt \rightarrow tq, t \in T$
Vorhersage (produce): $Xq \rightarrow x_n \dots x_1 q, X \rightarrow x_l \dots x_n \in P$	Reduktion (reduce): $x_1 \dots x_n q \rightarrow Xq, X \rightarrow x_1 \dots x_n \in P$

top-down Zerteiler

bottom-up Zerteiler

vom Startsymbol zum Wort

vom Wort zum Startsymbol

Anmerkung: Der Zustand ist bedeutungslos.

### 3.3 Herleitung der LR-Zerteiler

- Gegeben Grammatik  $G=(\Sigma, N, P, Z)$ ,  $V = \Sigma \cup N$ , konstruiere indeterministischen Kellerautomat mit genau einem Zustand  $q$ , angesetzt auf Eingabe  $x$
- Mache Kellerautomat deterministisch durch Hinzunahme von Rechtskontext, also Reduktion  $x_1 \dots x_n q x' \rightarrow Xq x'$ ,  $x'$  Anfang des unverarbeiteten Eingaberest
- **Deterministisch machen geht nur für eingeschränkte Grammatikklassen**

### 3.3 LR-Grammatiken

**Ziel:**

- alle deterministisch zerteilbaren kfG charakterisieren ( $LL(k)$  ist stark eingeschränkt)
- Rechtsableitung konstruieren

Endgültige Definition von D.E. Knuth 1966:

Eine kf-Grammatik heißt eine  $LR(k)$ -Grammatik, wenn für alle Paare von Ableitungen

$$Z \Rightarrow_R \mu A \omega \Rightarrow \mu \chi \omega \quad \mu \in V^*, \omega \in \Sigma^*, A \rightarrow \chi \in P$$

$$Z \Rightarrow_R \mu B \omega' \Rightarrow \mu \gamma \omega' \quad \mu' \in V^*, \omega' \in \Sigma^*, B \rightarrow \gamma \in P$$

gilt

$$(|\mu \chi| + k) : \mu \chi \omega = (|\mu \gamma| + k) : \mu \gamma \omega' \Rightarrow \mu = \mu', A = B, \chi = \gamma.$$

- **Probleme:** LR-Automat nur automatisch generierbar, sehr große Tabellen, Test der Eigenschaft nur durch Konstruktion möglich
- **In der Praxis** Beschränkung auf Unterklassen von  $LR(1)$  mit  $LR(0)$ -Zustandsmenge.

### 3.3 Reduktionsklassen

Unter welchen Bedingungen soll der LR-Automat schiften/reduzieren?

maximal verfügbare Information:

**Reduktionsklasse**  $\{(Kellerinhalt, Eingaberest)\}$   $R_0$  für Schift und  $R_p$  für Produktionen  $A_p \rightarrow y_p, p=1, \dots, n$ :

$$R_0 = \{(r, s) \mid r = r''r', s = s's'' \text{ mit } Z \Rightarrow_R r''As'', A \Rightarrow_{R'} r's', s' \neq \varepsilon\}$$

$$R_p = \{(r, s) \mid r = r''y_p, \text{ mit } Z \Rightarrow_R r''A_p s, A_p \rightarrow y_p \in P\}$$

$R_i \cap R_j = \emptyset$  für  $i \neq j$  bedeutet: Schiften bzw. Reduzieren mit Produktion  $p$  kann stets eindeutig entschieden werden. Jeder Satz besitzt eindeutige Rechtsableitung und eindeutigen Strukturbaum.

Grammatik ist eindeutig.

**Leider:** Eindeutigkeit von kf Grammatiken nicht entscheidbar, d.h.  $R_i \cap R_j = \emptyset$  nicht algorithmisch entscheidbar. Problem ist der unbeschränkt lange Eingaberest.



### 3.3 Textmengen (Wiederholung)

$$k : x = x\#, \quad \text{falls } |x| < k$$

$$k : x = x_1 \dots x_k, \quad \text{falls } x = x_1 \dots x_m \wedge m \geq k$$

$$\text{Anf}_k(x) = \{u \mid \exists y \in \Sigma^* \text{ so daß } x \Rightarrow^* y, u = k : y\}$$

$$\text{Anf}'_k(x) = \{u \mid u \in \text{Anf}_k(x) \wedge \neg \exists A \in N, y \in \Sigma^* \text{ so daß } x \Rightarrow_R Auy \Rightarrow uy\}$$

$$\text{Folge}_k(x) = \{u \mid \exists y \in V^* \text{ so daß } Z \Rightarrow^* mxy, u \in \text{Anf}_k(y)\}$$



### 3.3 Warum ist Anf' nötig: Beispiel

$L(G) = \{cb, b\}$  und

$$G = \quad 1 : Z \rightarrow S, \quad 2 : S \rightarrow Ab,$$

$$3 : A \rightarrow c, \quad 4 : A \rightarrow \varepsilon$$

$$R_0 = \{(\varepsilon, cb\#), (A, b\#), (\varepsilon, b\#)\}, \quad R_1 = \{(S, \#)\},$$

$$R_2 = \{(Ab, \#)\}, \quad R_3 = \{(c, b\#)\},$$

$$R_4 = \{(\varepsilon, b\#)\}.$$

Rechtsableitung  $Z \Rightarrow_R S \Rightarrow_R Ab \Rightarrow_R b$ :

$\varepsilon$  auf  $A$  reduzieren, bevor das Zeichen  $b$  geschifft wird. **Ohne**

Unterscheidung zwischen  $\Rightarrow_{R'}$  und  $\Rightarrow_R$ :

$(\varepsilon, b\#)$  gehört zu  $R_0$ , also **Sackgasse**



### 3.3 k-Kellerklassen

Idee: Beschränke den betrachteten Eingaberest auf  $k \geq 0$  Zeichen:

$k$ -Kellerklasse  $K_p^k, p=0, \dots, n$  definiert durch

$$K_p^k = \{(r, \underline{s}) \mid \exists (r, s) \in R_p \text{ mit } \underline{s} = k:s\}$$

Vergleich mit LR-Definition: Mit

$$Z \Rightarrow_R \mu A \omega \Rightarrow \mu \chi \omega \quad \mu \in V^*, \omega \in T^*, A \rightarrow \chi \in P$$

$$Z \Rightarrow_R \mu' B \omega' \Rightarrow \mu' \gamma \omega' \quad \mu' \in V^*, \omega' \in T^*, B \rightarrow \gamma \in P$$

gilt

$$(|\mu \chi| + k) : \mu \chi \omega = (|\mu' \gamma| + k) : \mu' \gamma \omega' \Rightarrow \mu = \mu', A = B, \chi = \gamma.$$

**Satz:** Eine Grammatik ist genau dann LR( $k$ ), wenn die  $k$ -Kellerklassen paarweise disjunkt sind. Eine Grammatik  $G$  ist genau dann deterministisch zerteilbar, wenn es ein  $k \geq 0$  gibt, so daß  $G$  LR( $k$ ) ist.





### 3.3 Situationen (Wiederholung)

Ziel: bei Prüfung der Anwendbarkeit von Regeln  $sqx \rightarrow s'q'x'$   
 $sq$  in **einem** Zustandssymbol codieren (Prüfung mehrerer Symbole im Keller vermeiden)

Lösungsidee:

- bei  $LL$  und  $LR$  ist  $s$  (Teil einer) rechten bzw. linken Seite einer Produktion  $X \rightarrow x_1 \dots x_n$
- Übergang  $qt \rightarrow q't$  sind nur zulässig, wenn in der Produktion ein Terminalzeichen  $t$  ansteht,  $x_1 \dots x_n = x'x''$  und  $x'' = tx''$
- also: ersetze  $sq$  durch **Situation**  $[X \rightarrow x' \bullet x''; x]$ , die durch den Punkt anzeigt, wie weit die Produktion abgearbeitet ist.
- Situationen  $[X \rightarrow \bullet x''; x]$  oder  $[X \rightarrow x' \bullet; x]$  sind erlaubt und notwendig.
- Verwende Situationen als Zustände **und** als Kellersymbole



### 3.3 Entscheidungsverfahren, $k$ -Kellerklassen

**Satz** (Büchi):

Die  $k$ -Kellerklassen sind regulär.

**Beweis** durch Angabe einer Grammatik  $G_i^k$  für  $K_i^k$ ,  
 $i=0, \dots, n$ .

Dabei spielt die  $k$ -Vorschau keine wichtige Rolle, da es nur endlich viele Kombinationen von  $k$  Zeichen gibt.

Eigenschaften von  $G_i^k$

- Situationen als Nichtterminale, Symbole  $t \in V$  als Terminale
- Ziel  $[Z \rightarrow S \bullet; \#]$



### 3.3 Reguläre Grammatik für $k$ -Kellerklassen

Situationen  $[X \rightarrow x \bullet ty; \omega]$  als Nichtterminale,

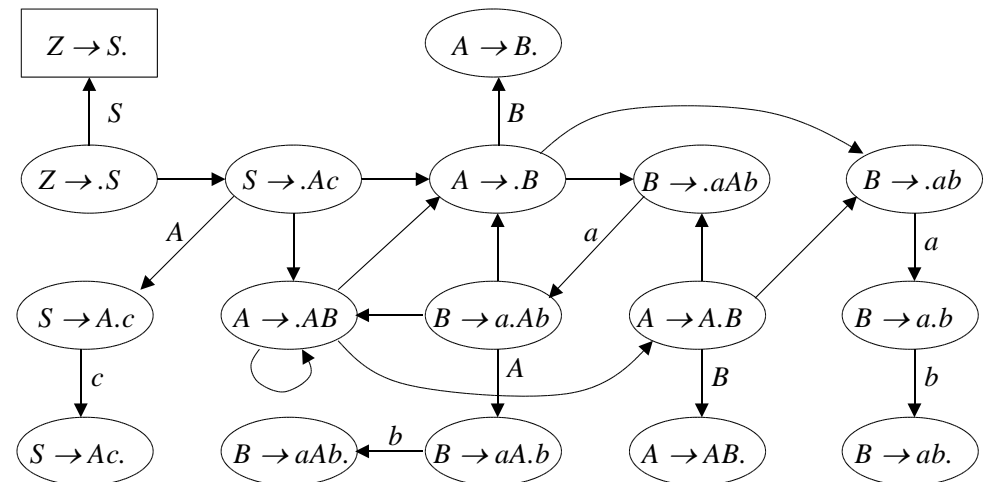
- **Übergänge:** mit  $t \in V$  in Folgezustand  
 $P' = \{[X \rightarrow x \bullet ty; \omega] \rightarrow t [X \rightarrow xt \bullet y; \omega] \mid t \in V\}$
- mit  $B \in N$  im Keller spontan Unterproduktion  $B \rightarrow v$  beginnen  
 $P'' = \{[X \rightarrow x \bullet By; \omega] \rightarrow [B \rightarrow \bullet v; \tau] \mid B \rightarrow v \in P, \tau \in \text{Anf}_k'(y\omega)\}$
- mit  $t \in V$  in Endzustand für  $K_0^k$ , wenn Vorschau stimmt  
 $P_0 = \{[X \rightarrow x \bullet y; \omega] \rightarrow \&\tau \mid y \neq \epsilon, \tau \in \text{Anf}_k'(y\omega)\}$
- von  $[X_p \rightarrow x_p \bullet; \omega]$  in Endzustand für  $K_p^k$ , wenn Vorschau stimmt  
 $P_p = \{[X_p \rightarrow x_p \bullet; \omega] \rightarrow \&\omega\}, p=1, \dots, n$

Produktionen für  $G_i^k: P' \cup P'' \cup P_p, i=0, \dots, n$ , also alle Grammatiken gleich außer Abschluß einer Ableitung (Übergang Endzustand)



### 3.3 Nicht-deterministischer, endlicher Automat

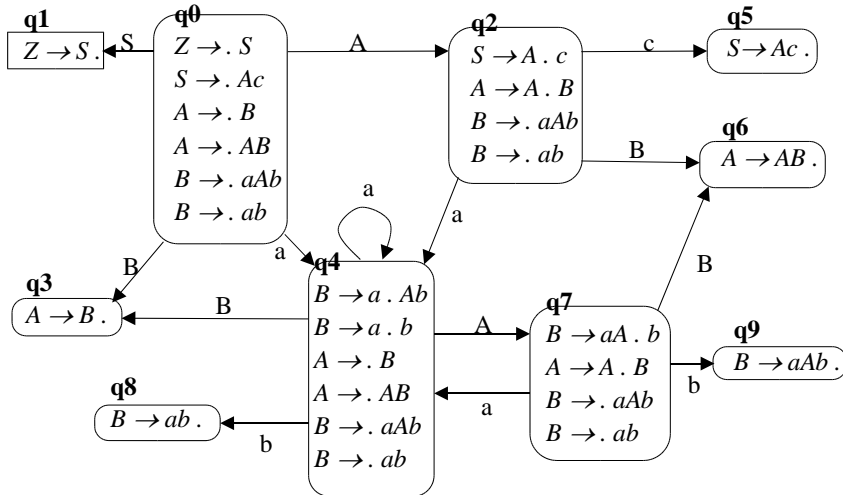
$Z \rightarrow S, S \rightarrow Ac, A \rightarrow AB, A \rightarrow B, B \rightarrow aAb, B \rightarrow ab$



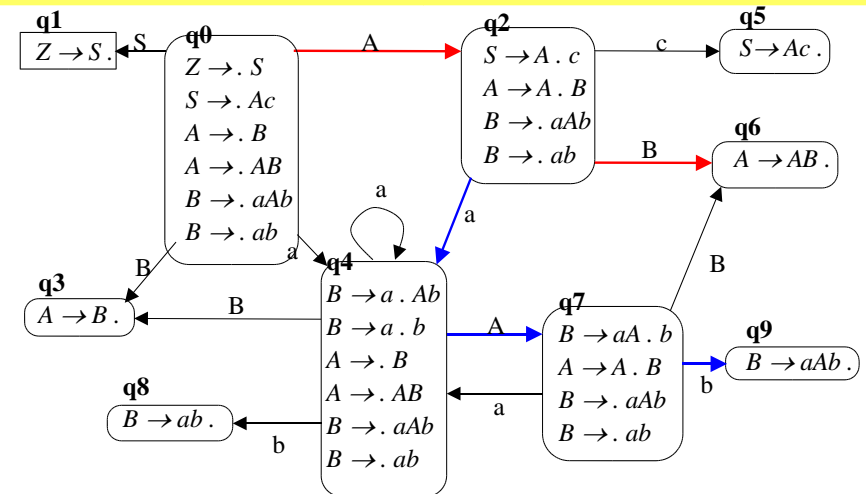


### 3.3 Charakteristischer Automat

$Z \rightarrow S, S \rightarrow Ac, A \rightarrow AB, A \rightarrow B, B \rightarrow aAb, B \rightarrow ab$



### 3.3 Charakteristischer Automat als Kellerautomat



Ableitung  $Z \Rightarrow S \Rightarrow AB \Rightarrow AaAb \Rightarrow \dots$  Bei Reduktion  $B \rightarrow aAb$  in  $q_9$  gehe  $3 = |aAb|$  Schritte zurück und dann einen Schritt nach  $q_6$  mit  $B$

### 3.3 Konstruktion des charakteristischen Automaten

Entscheidungsverfahren nach Büchi:

- Aus den Grammatiken deterministische endliche Automaten machen: Anwendung der Teilmengenkonstruktion und Hüllenbildung:
  - Transitiv Hülle eines Nichtterminals:  $H(A) = \{A\} \cup \{B \mid C \rightarrow B \in P, C \in H(A)\}$
  - angewandt auf Mengen  $M$  von Situationen:  $H(M) = M \cup \{[B \rightarrow \bullet b; t] \mid \exists [X \rightarrow \mu \bullet B \gamma; \omega] \in H(M); B \rightarrow \beta \in P, t \in \text{Anf}_k(\gamma \omega)\}$
- Kellerinhalt und Vorschau durch alle Automaten schicken
- Endzustand bestimmt, welche Aktion (Schift, Reduktion mit Produktion  $p$ ) ausgeführt wird, falls Kellerklassen disjunkt

Erkenntnis: alle Automaten sind gleich außer Übergang in Endzustand, also nur ein **charakteristischer Automat** mit verschiedenen Endzuständen

### 3.3 Charakteristischer Automat als LR-Kellerautomat

- charakteristischen Automaten mit Keller ergänzen
  - statt Zeichen  $t \in V$  Zustände, d.h. Mengen von Situationen, als Kellersymbole
- oberstes Kellersymbol ist laufender Zustand
- Beginn mit  $H(\{[Z \rightarrow \bullet S; \#]\})$
- Übergang in Endzustand für Schiften mit Schiften der Folgesituation in den Keller zusammenlegen, d.h. die Endzustände für  $K_0^k$  streichen. Folgezustand von  $q$  bei Zeichen  $t$ :
  - $\text{basis}(q,t) = \{[X \rightarrow \mu t \bullet \gamma; \Omega] \mid [X \rightarrow \mu \bullet t \gamma; \Omega] \in q\}$ .
  - $\text{next}(q,t) = H(\text{basis}(q,t))$ , wenn  $\text{basis}(q,t) \neq \emptyset$
- Nach Erreichen Endzustand für  $K_p^k, p=1, \dots, n, p: B \rightarrow x_1 \dots x_m : m$  Situationen aus Keller streichen, Zustandsübergang mit  $B$
- spart die Ableitung des Kelleraufbaus, die unverändert ist

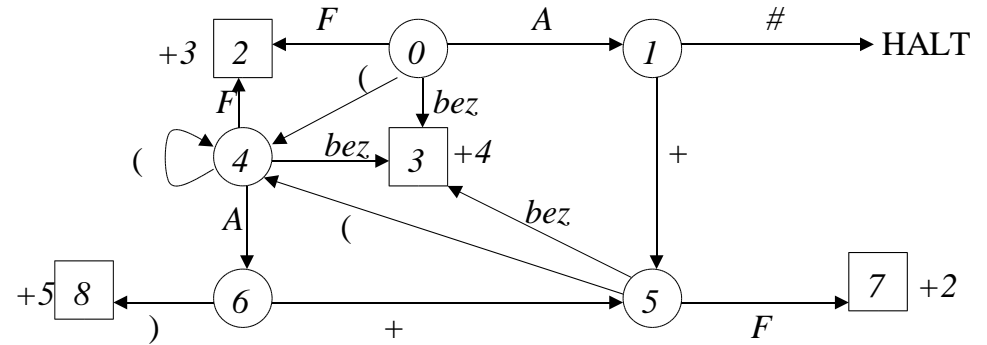
### 3.3 Übergangsfunktion des LR-Automaten

- Für  $v \in V$ :
- $$f(q, v) = \begin{cases} \text{next}(q, v), & \text{wenn } \text{basis}(q, v) \neq \emptyset, \text{ (schiften)} \\ X \rightarrow x, & \text{wenn } [X \rightarrow x \bullet; \Omega] \in q, v \in \Omega, \text{ (reduzieren)} \\ \text{HALT}, & \text{wenn } v = \# \text{ und } [Z \rightarrow S \bullet; \{\#\}] \in q \\ \text{FEHLER}, & \text{sonst} \end{cases}$$
- Zustand  $q$  heißt **inadäquat**, wenn Übergangsfunktion  $f(q, v)$  für irgendein  $v$  nicht eindeutig definiert. Zwei Möglichkeiten:
  - Schift-Reduktionskonflikt: sowohl schiften als auch reduzieren möglich
  - Reduktions-Reduktionskonflikt: Reduktion mit zwei verschiedenen Produktionen möglich
  - Eine kfg  $G$  ist genau dann  $LR(k)$ -Grammatik, wenn der charakteristische Automat keine inadäquaten Zustände besitzt.

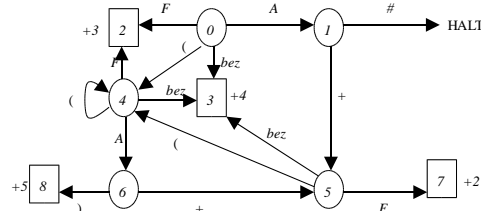


### 3.3 Vereinfachter charakteristischer Automat, $k = 1$

- (1)  $Z \rightarrow A$
- (2)  $A \rightarrow A + F$
- (3)  $A \rightarrow F$
- (4)  $F \rightarrow \text{bez}$
- (5)  $F \rightarrow (A)$



vereinfacht: Zustände mit unterschiedlicher Vorschau zusammengelegt



Keller	Erkannt	Eingabe	Bemerkungen
0		$a+(b+c)\#$	
0 3	$a$	$+(b+c)\#$	Red. $F \rightarrow \text{bez}$
0 0	$F$	$+(b+c)\#$	Red. $A \rightarrow F$
0 2	$F$	$+(b+c)\#$	
0 1	$A$	$+(b+c)\#$	
0 1 5	$A +$	$(b+c)\#$	
0 1 5 4	$A + ($	$b+c)\#$	
0 1 5 4 3	$A + (b$	$+c)\#$	Reduktion $F \rightarrow \text{bez}$
0 1 5 4	$A + (F$	$+c)\#$	Reduktion $A \rightarrow F$
0 1 5 4 2	$A + (F$	$+c)\#$	
0 1 5 4	$A + (A$	$+c)\#$	
0 1 5 4 6	$A + (A$	$+c)\#$	
0 1 5 4 6 5	$A + (A +$	$c)\#$	
0 1 5 4 6 5 3	$A + (A + c$	$)\#$	Reduktion $F \rightarrow \text{bez}$
0 1 5 4 6 5	$A + (A + F$	$)\#$	Reduktion $A \rightarrow A + F$
0 1 5 4 6 5 7	$A + (A + F$	$)\#$	
0 1 5 4	$A + (A$	$)\#$	
0 1 5 4 6	$A + (A$	$)\#$	
0 1 5 4 6 8	$A + (A$	$)\#$	
0 1 5	$A + F$	$\#$	Reduktion $F \rightarrow (A)$
0 1 5 7	$A + F$	$\#$	Reduktion $A \rightarrow A + F$
0	$A$	$\#$	
0 1	$A$	$\#$	
0 1	$A \#$	$\#$	HALT

rot: nach Kürzen des Kellers

### 3.3 Konstruktion LR(1) Kellerautomat

- Initial  $Q = \{q_0\}$ , mit  $q_0 = H(\{[Z \rightarrow \bullet S; \#]\})$ .
- Sei  $q \in Q$  noch nicht betrachtet. Führe Schritte (3)-(5) für jedes  $v \in V$  aus.
- Wenn  $\text{basis}(q, v) \neq \emptyset$ , setze  $\text{next}(q, v) = H(\text{basis}(q, v))$ .  $Q := Q \cup \{q' = \text{next}(q, v)\}$ .
- Wenn  $\text{basis}(q, v) \neq \emptyset$  und  $v \in T$ , für  $k \leq 1$  setze  $R := R \cup \{qv \rightarrow qq'\}$ , sonst setze  $R := R \cup \{qvt \rightarrow qq't \mid [X \rightarrow x \bullet v\gamma; \omega] \in q, t \in \text{Anf}_{k-1}(\gamma\omega)\}$ .
- Wiederhole Schritt (2) bis alle  $q \in Q$  betrachtet sind.
- Für jedes  $q \in Q$  und  $[X \rightarrow x \bullet; \omega] \in q$  mit  $x = x_1 \dots x_n$  setze
 
$$R := R \cup \{q_1 \dots q_n q \omega \rightarrow q_1 q' \omega \mid [X \rightarrow x \bullet; \omega] \in q_1, q_{i+1} = \text{next}(q_i, x_i) \ (i=1, \dots, n-1), q = \text{next}(q_n, x_n), q' = \text{next}(q, X)\}$$



### 3.3 Beispiel: LR(2) Kellerautomat

Grammatik	Zustände	Übergang	Zustände	Übergang
(1) $Z \rightarrow X$	$q_0$ : $[Z \rightarrow \bullet X; \#]$	$b, c$	$q_4$ : $[Y \rightarrow c \bullet; \#]$	red #
(2) $X \rightarrow Y$	$[X \rightarrow \bullet Y; \#]$		$[Y \rightarrow c \bullet a; \#]$	a#
(3) $X \rightarrow bYa$	$[X \rightarrow \bullet bYa; \#]$		$q_5$ : $[X \rightarrow bY \bullet a; \#]$	a#
(4) $Y \rightarrow c$	$[Y \rightarrow \bullet c; \#]$		$q_6$ : $[Y \rightarrow c \bullet a; \#]$	red a#
(5) $Y \rightarrow ca$	$[Y \rightarrow \bullet ca; \#]$		$[Y \rightarrow c \bullet a; a\#]$	aa
	$q_1$ : $[Z \rightarrow X \bullet; \#]$	HALT	$q_7$ : $[Y \rightarrow ca \bullet; \#]$	red #
	$q_2$ : $[X \rightarrow Y \bullet; \#]$	red #	$q_8$ : $[X \rightarrow bYa \bullet; \#]$	red #
	$q_3$ : $[X \rightarrow b \bullet Ya; \#]$	$c$	$q_9$ : $[Y \rightarrow ca \bullet; a\#]$	red a#
	$[Y \rightarrow \bullet c; a\#]$			
	$[Y \rightarrow \bullet ca; a\#]$			

Regeln  $R = \{q_0bc \rightarrow q_0q_3c, q_0c\# \rightarrow q_0q_4\#, q_0ca \rightarrow q_0q_4a, q_3ca \rightarrow q_3q_6a, q_4a\# \rightarrow q_4q_7\#, q_5a\# \rightarrow q_5q_8\#, q_6aa \rightarrow q_6q_9a, q_0q_2\# \rightarrow q_0q_1\#, q_0q_4\# \rightarrow q_0q_2\#, q_3q_6a\# \rightarrow q_3q_5a\#, q_3q_4q_7\# \rightarrow q_0q_2\#, q_0q_3q_5q_8\# \rightarrow q_0q_1\#, q_3q_6q_9\# \rightarrow q_3q_5\#\}$



### 3.3 SLR(k)-Grammatiken

Problem: Wegen Unterscheidung verschiedener Rechtskontexte hat der  $LR(k)$ -Automat bereits für  $k=1$  sehr viele Zustände, verglichen mit  $LR(0)$   
 Beispiel: für Ausdrucksgrammatik mit „+“ 17 statt 9 für  $k=0$ .

**SLR(k)-Grammatik (simple LR(k)):** Eine Grammatik heißt  $SLR(k)$ , wenn sie  $LR(0)$  ist oder die modifizierte Übergangsfunktion

$$f(q, v) = \begin{cases} next(q, v) & \text{wenn } basis(q, v) \neq \emptyset, \text{ (schiften)} \\ X \rightarrow x & \text{wenn } [X \rightarrow x \bullet] \in q, v \in Folge_k(X), \text{ (reduzieren)} \\ \text{HALT} & \text{wenn } v = \# \text{ und } [Z \rightarrow S \bullet] \in q \\ \text{FEHLER} & \text{sonst} \end{cases}$$

keine inadäquaten Zustände liefert.

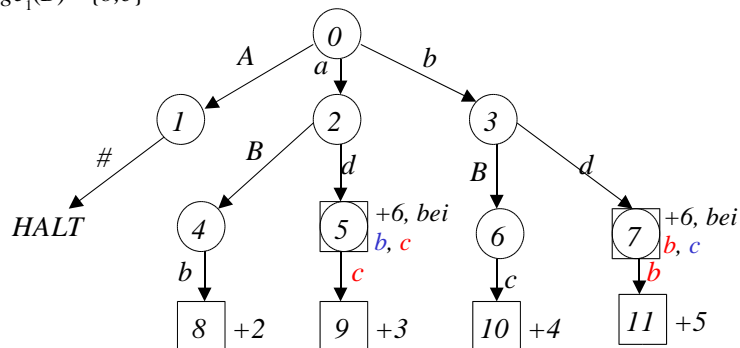
Problem:  $SLR(k)$  genügt für Ausdrucksgrammatiken, berücksichtigt aber Rechtskontext zu wenig.

Achtung: Keine Analogie zu LL vs. SLL (LL ohne Keller).



### 3.3.1 nicht SLR, aber LALR

- (1)  $Z \rightarrow A$       (2)  $A \rightarrow aBb$       (3)  $A \rightarrow adc$   
 (4)  $A \rightarrow bBc$       (5)  $A \rightarrow bdb$       (6)  $B \rightarrow d$   
 $Folge_1(B) = \{b, c\}$



praktisches Beispiel (Algol 60): Unterscheide ;abc: und [abc:



### Kapitel 3: Zerteilung

- Theoretische Grundlage: Kontextfreie Grammatiken
  - Notation
  - Konkrete und abstrakte Syntax
  - Rekursiver Abstieg
  - Kellerautomaten
  - Elimination von Linksrekursion und  $\epsilon$ -Produktionen
  - Systematische Zerteilerkonstruktion: LL-, LR-Grammatiken
- LL- und SLL-Grammatiken
- LR-, SLR-Grammatiken
  - 3.1 LALR-Konstruktion
  - 3.2 Optimierungen und Komplexität
- Fehlerbehandlung



### 3.3.1 LALR(k)-Grammatiken

Sei  $kern(q) = \{ [X \rightarrow \mu \cdot \gamma] \mid [X \rightarrow \mu \cdot \gamma ; \Omega] \in q \}$ .

- Eine Grammatik heißt **LALR(k) (look ahead LR(k))**, wenn es keine inadäquaten Zustände gibt, falls man im LR(k)-Automaten alle Zustände  $q, q'$  mit  $kern(q) = kern(q')$  zusammenlegt.
- **Satz:** Jeder SLR(k)- oder LALR(k)-Automat hat die gleiche Anzahl von Zuständen wie der LR(0)-Automat zur gleichen Grammatik.
- Der Unterschied zwischen LALR(k) und LR(k) ist praktisch unerheblich. Alle verbreiteten LR-Zerteilergeneratoren (*yacc, pgs, lalr, bison, ...*) konstruieren LALR(1)-Automaten.



### 3.3.1 Konstruktion LALR(1) Kellerautomat

Erweitern von Zuständen mit **gleichem Kern:**

$$erweitere(q, q') = \{ [ \dots ; \Omega \cup \Delta ] \mid [ \dots ; \Omega ] \in q, [ \dots ; \Delta ] \in q' \}.$$

1. Initial  $Q = \{q_0\}$ , mit  $q_0 = H(\{ [Z \rightarrow \cdot S ; \#] \})$ .
2. Sei  $q \in Q$  noch nicht betrachtet. Für alle  $v \in V$  führe Schritt (3) aus.
3. Wenn  $basis(q, v) \neq \emptyset$ , berechne  $\underline{q} = H(basis(q, v))$ .
  - 3a. Wenn es ein  $q' \in Q$  mit  $kern(q') = kern(\underline{q})$  gibt, so setze  $next(q, v) = erweitere(\underline{q}, q')$ ; wenn  $next(q, v) \neq q'$ , ersetze  $q'$  durch  $next(q, v)$  und markiere  $q'$  als nicht betrachtet.
  - 3b. Wenn es kein  $q' \in Q$  mit  $kern(q') = kern(\underline{q})$  gibt, so setze  $next(q, v) = \underline{q}$ , nimm  $next(q, v)$  in  $Q$  auf und markiere es als nicht betrachtet.
4. Wiederhole Schritt (2) bis alle  $q \in Q$  betrachtet sind.
5. Für jedes  $q \in Q$  und  $v \in V$ , berechne  $f(q, v)$ .



### 3.3.1 Wiederholung Übergangsfunktion

Für  $v \in V$ :

$$f(q, v) = \begin{cases} next(q, v), & \text{wenn } basis(q, v) \neq \emptyset, & \text{(schieben)} \\ X \rightarrow x, & \text{wenn } [X \rightarrow x \cdot ; \Omega] \in q, v \in \Omega, & \text{(reduzieren)} \\ \text{HALT,} & \text{wenn } v = \# \text{ und } [Z \rightarrow S \cdot ; \{ \# \}] \in q \\ \text{FEHLER,} & \text{sonst} \end{cases}$$

### 3.3.1 Konstruktion am Beispiel

- |     |                       |     |                   |
|-----|-----------------------|-----|-------------------|
| (1) | $Z \rightarrow A$     | (3) | $A \rightarrow T$ |
| (2) | $A \rightarrow A + T$ | (5) | $T \rightarrow F$ |
| (4) | $T \rightarrow T * F$ |     |                   |
| (6) | $F \rightarrow bez$   |     |                   |
| (7) | $F \rightarrow (A)$   |     |                   |



### 3.3.1 Anfangszustand

Zustand $q_0$		Übergang	
Basis	Hülle	mit	in neue Basis
$[Z \rightarrow \bullet A; \{\#\}]$		$A$	$\{[Z \rightarrow A \bullet; \{\#\}]\}$
	$[A \rightarrow \bullet A + T; \{+\#\}]$		$[A \rightarrow A \bullet + T; \{+\#\}]\}$
	$[A \rightarrow \bullet T; \{+\#\}]$	$T$	$\{[A \rightarrow T \bullet; \{+\#\}]\}$
	$[T \rightarrow \bullet T * F; \{*\#\}]$		$[T \rightarrow T \bullet * F; \{*\#\}]\}$
	$[T \rightarrow \bullet F; \{*\#\}]$	$F$	$\{[T \rightarrow F \bullet; \{*\#\}]\}$
	$[F \rightarrow \bullet bez; \{*\#\}]$	$bez$	$\{[F \rightarrow bez \bullet; \{*\#\}]\}$
	$[F \rightarrow \bullet (A); \{*\#\}]$	$($	$\{[F \rightarrow ( \bullet A); \{*\#\}]\}$



### 3.3.1 Zustand 1

Zustand $q_1$		Übergang	
Basis	Hülle	mit	in neue Basis
$[Z \rightarrow A \bullet; \{\#\}]$		$\#$	Reduktion, HALT
$[A \rightarrow A \bullet + T; \{+\#\}]$		$+$	$\{[A \rightarrow A + \bullet T; \{+\#\}]\}$



### 3.3.1 Zustand 2

Zustand $q_2$		Übergang	
Basis	Hülle	mit	in neue Basis
$[A \rightarrow T \bullet; \{+\#\}]$			Reduktion bei $+, \#$
$[T \rightarrow T \bullet * F; \{*\#\}]$		$*$	$\{[T \rightarrow T * \bullet F; \{*\#\}]\}$



### 3.3.1 Zustand 3

Zustand $q_3$		Übergang	
Basis	Hülle	mit	in neue Basis
$[T \rightarrow F \bullet; \{*\#\}]$			Reduktion bei $*, +, \#$



### 3.3.1 Zustand 4

Zustand $q_4$		Übergang	
Basis	Hülle	mit	in neue Basis
$[F \rightarrow bez \bullet; \{ *+ \# \}]$			Reduktion bei *,+,#



### 3.3.1 Zustand 5

Zustand $q_5$		Übergang	
Basis	Hülle	mit	in neue Basis
$[F \rightarrow (\bullet A); \{ *+ \# \}]$		A	$\{ [F \rightarrow (A \bullet); \{ *+ \# \}] \}$
	$[A \rightarrow \bullet A + T; \{ + \}]$		$[A \rightarrow A \bullet + T; \{ + \}]$
	$[A \rightarrow \bullet T; \{ + \}]$	T	$\{ [A \rightarrow T \bullet; \{ + \}] \}$
	$[T \rightarrow \bullet T * F; \{ *+ \}]$		$[T \rightarrow T \bullet * F; \{ *+ \}]$
	$[T \rightarrow \bullet F; \{ *+ \}]$	F	$\{ [T \rightarrow F \bullet; \{ *+ \}] \}$
	$[F \rightarrow \bullet bez; \{ *+ \}]$	bez	$\{ [F \rightarrow bez \bullet; \{ *+ \}] \}$
	$[F \rightarrow \bullet (A); \{ *+ \}]$	(	$\{ [F \rightarrow (\bullet A); \{ *+ \}] \}$

Die Hülle blau gekennzeichnete Basen erweitert vorhandene Zustände.



### 3.3.1 Erweitere Zustand 2

Zustand $q_2$		Übergang	
Basis	Hülle	mit	in neue Basis
$[A \rightarrow T \bullet; \{ + \# \}]$			Reduktion bei +, #
$[T \rightarrow T \bullet * F; \{ *+ \# \}]$		*	$\{ [T \rightarrow T * \bullet F; \{ *+ \# \}] \}$

$[A \rightarrow T \bullet; \{ + \}]$   
 $[T \rightarrow T \bullet * F; \{ *+ \}]$

Neuer Zustand  $q_2$

$[A \rightarrow T \bullet; \{ + \# \}]$   
 $[T \rightarrow T \bullet * F; \{ *+ \# \}]$

Reduktion bei +, #, )

\*



### 3.3.1 Erweitere Zustand 3

Zustand $q_3$		Übergang	
Basis	Hülle	mit	in neue Basis
$[T \rightarrow F \bullet; \{ *+ \# \}]$			Reduktion bei *,+,#

$[T \rightarrow F \bullet; \{ *+ \}]$

Neuer Zustand  $q_3$

$[T \rightarrow F \bullet; \{ *+ \# \}]$

Reduktion bei \*,+,#,) )



### 3.3.1 Erweitere Zustand 4

Zustand $q_4$		Übergang	
Basis	Hülle	mit	in neue Basis
$[F \rightarrow bez \bullet; \{ * + \# \}]$			Reduktion bei *, +, #
$[F \rightarrow bez \bullet; \{ * + \}]$			
Neuer Zustand $q_4$			
$[F \rightarrow bez \bullet; \{ * + \# \}]$			Reduktion bei *, +, #, )



### 3.3.1 Erweitere Zustand 5

Zustand $q_5$		Übergang	
Basis	Hülle	mit	in neue Basis
$[F \rightarrow (\bullet A); \{ * + \# \}]$		A	$\{ [F \rightarrow (A \bullet); \{ * + \# \}] \}$
	$[A \rightarrow \bullet A + T; \{ + \}]$		$[A \rightarrow A \bullet + T; \{ + \}]$
	$[A \rightarrow \bullet T; \{ + \}]$	T	$\{ [A \rightarrow T \bullet; \{ + \}] \}$
	$[T \rightarrow \bullet T * F; \{ * + \}]$		$[T \rightarrow T \bullet * F; \{ * + \}]$
	$[T \rightarrow \bullet F; \{ * + \}]$	F	$\{ [T \rightarrow F \bullet; \{ * + \}] \}$
	$[F \rightarrow \bullet bez; \{ * + \}]$	bez	$\{ [F \rightarrow bez \bullet; \{ * + \}] \}$
	$[F \rightarrow \bullet (A); \{ * + \}]$	(	$\{ [F \rightarrow (\bullet A); \{ * + \}] \}$
$[F \rightarrow (\bullet A); \{ * + \}]$			
Neuer Zustand enthält ) in der Folgemenge:			
$[F \rightarrow (\bullet A); \{ * + \# \}]$		A	$\{ [F \rightarrow (A \bullet); \{ * + \# \}] \}$
$[A \rightarrow \bullet A + T; \{ + \}]$			$[A \rightarrow A \bullet + T; \{ + \}]$
...		...	



### 3.3.1 Zustand 6

Zustand $q_6$		Übergang	
Basis	Hülle	mit	in neue Basis
$[A \rightarrow A + \bullet T; \{ + \# \}]$		T	$\{ [A \rightarrow A + T \bullet; \{ + \# \}] \}$
$\{ * + \# \}]$	$[T \rightarrow \bullet T * F; \{ * + \# \}]$		$[T \rightarrow T \bullet * F; \{ * + \# \}]$
	$[T \rightarrow \bullet F; \{ * + \# \}]$	F	$\{ [T \rightarrow F \bullet; \{ * + \# \}] \}$
	$[F \rightarrow \bullet bez; \{ * + \# \}]$	bez	$\{ [F \rightarrow bez \bullet; \{ * + \# \}] \}$
	$[F \rightarrow \bullet (A); \{ * + \# \}]$	(	$\{ [F \rightarrow (\bullet A); \{ * + \# \}] \}$

Die Hülle rot gekennzeichnete Basen ergibt keine Erweiterung vorhandener Zustände.



### 3.3.1 Zustand 7

Zustand $q_7$		Übergang	
Basis	Hülle	mit	in neue Basis
$[T \rightarrow T * \bullet F; \{ * + \# \}]$		F	$\{ [T \rightarrow T * F \bullet; \{ * + \# \}] \}$
	$[F \rightarrow \bullet bez; \{ * + \# \}]$	bez	$\{ [F \rightarrow bez \bullet; \{ * + \# \}] \}$
	$[F \rightarrow \bullet (A); \{ * + \# \}]$	(	$\{ [F \rightarrow (\bullet A); \{ * + \# \}] \}$



### 3.3.1 Zustand 8

Zustand $q_8$		Übergang	
Basis	Hülle	mit	in neue Basis
$[F \rightarrow (A\bullet); \{*\#\}]$		)	$\{[F \rightarrow (A)\bullet]; \{*\#\}]\}$
$[A \rightarrow A\bullet+T; \{+\}]$		+	$\{[A \rightarrow A+T; \{+\}]\}$



### 3.3.1 Erweitere Zustand 6

Zustand $q_6$		Übergang	
Basis	Hülle	mit	in neue Basis
$[A \rightarrow A+\bullet T; \{+\#}]$		$T$	$\{[A \rightarrow A+T\bullet; \{+\#}]\}$
	$[T \rightarrow \bullet T^*F; \{*\#\}]$		$[T \rightarrow T^*F; \{*\#\}]$
	$[T \rightarrow \bullet F; \{*\#\}]$	$F$	$\{[T \rightarrow F\bullet; \{*\#\}]\}$
	$[F \rightarrow \bullet bez; \{*\#\}]$	$bez$	$\{[F \rightarrow bez\bullet; \{*\#\}]\}$
	$[F \rightarrow \bullet(A); \{*\#\}]$	(	$\{[F \rightarrow (\bullet A); \{*\#\}]\}$
$[A \rightarrow A+\bullet T; \{+\}]$			
Neuer Zustand enthält ) in der Folgemenge:			
$[A \rightarrow A+\bullet T; \{+\#}]$		$T$	$\{[A \rightarrow A+T\bullet; \{+\#}]\}$
	$[T \rightarrow \bullet T^*F; \{*\#\}]$		$[T \rightarrow T^*F; \{*\#\}]$
	$[T \rightarrow \bullet F; \{*\#\}]$	$F$	$\{[T \rightarrow F\bullet; \{*\#\}]\}$
	$[F \rightarrow \bullet bez; \{*\#\}]$	$bez$	$\{[F \rightarrow bez\bullet; \{*\#\}]\}$
	$[F \rightarrow \bullet(A); \{*\#\}]$	(	$\{[F \rightarrow (\bullet A); \{*\#\}]\}$



### 3.3.1 Zustand 9

Zustand $q_9$		Übergang	
Basis	Hülle	mit	in neue Basis
$[A \rightarrow A+T\bullet; \{+\#}]$			Reduktion bei +, #, )
$[T \rightarrow T\bullet^*F; \{*\#\}]$		*	$\{[T \rightarrow T^*\bullet F; \{*\#\}]\}$



### 3.3.1 Zustand 10

Zustand $q_{10}$		Übergang	
Basis	Hülle	mit	in neue Basis
$[T \rightarrow T^*F\bullet; \{*\#\}]$			Reduktion bei *, +, #, )





### 3.3.1 Zustand 11

Zustand $q_{11}$	Übergang		
Basis	Hülle	mit	in neue Basis
$[F \rightarrow (A)*; \{*, +, \# \}]$			Reduktion bei *, +, #, )



### 3.3.1 Übergangsfunktionen

$q_0$	$f(q_0, A) = q_1$ $f(q_0, T) = q_2$ $f(q_0, F) = q_3$ $f(q_0, bez) = q_4$ $f(q_0, () = q_5$
$q_1$	$f(q_1, \#) = \text{HALT}$ $f(q_1, +) = q_6$
$q_2$	$f(q_2, +\#) = \text{Reduziere}(A \rightarrow T)$ $f(q_2, *) = q_7$
$q_3$	$f(q_3, *+\#) = \text{Reduziere}(T \rightarrow F)$
$q_4$	$f(q_4, *+\#) = \text{Reduziere}(F \rightarrow bez)$
$q_5$	$f(q_5, A) = q_8$ $f(q_5, T) = q_2$ $f(q_5, F) = q_3$ $f(q_5, bez) = q_4$ $f(q_5, () = q_5$
$q_6$	$f(q_6, T) = q_9$ $f(q_6, F) = q_3$ $f(q_6, bez) = q_4$ $f(q_6, () = q_5$
$q_7$	$f(q_7, F) = q_{10}$ $f(q_7, bez) = q_4$ $f(q_7, () = q_5$
$q_8$	$f(q_8, ) = q_{11}$ $f(q_8, +) = q_6$
$q_9$	$f(q_9, +\#) = \text{Reduziere}(A \rightarrow A+T)$ $f(q_9, *) = q_7$
$q_{10}$	$f(q_{10}, *+\#) = \text{Reduziere}(T \rightarrow T*F)$
$q_{11}$	$f(q_{11}, *+\#) = \text{Reduziere}(F \rightarrow (A))$



### 3.3.1 Übergangstabelle

	bez	(	)	+	*	#	A	T	F
0	4	5	-	-	-	-	1	2	3
1	-	-	-	6	-	*			
2	-	-	+3	+3	7	+3			
3	+5	+5	+5	+5	+5	+5			
4	+6	+6	+6	+6	+6	+6			
5	4	5	-	-	-	-	8	2	3
6	4	5	-	-	-	-		9	3
7	4	5	-	-	-	-			10
8	-	-	11	6	-	-			
9	-	-	+2	+2	7	+2			
10	+4	+4	+4	+4	+4	+4			
11	+7	+7	+7	+7	+7	+7			



### 3.3.1 Fakten und Fragen zu LR

- Automat  $qt \rightarrow tq, t \in T, x_1 \dots x_n q \rightarrow Xq, X \rightarrow x_1 \dots x_n \in P$  deterministisch machen
  - dazu Reduktionsklassen bestimmen, Kellerklassen ableiten
  - Kellerklassen sind regulär, Produktionen mit Situationen als Nichtterminale
- charakteristischen Automaten herleiten: nicht-deterministischen Automaten in deterministischen überführen (Hüllenbildung)
- LR-Automat zu groß: SLR-Klasse (Vorschau: Folge<sub>k</sub>(A)) ungenügend
- LALR: Zustände des LR-Automaten mit gleichem Kern verschmelzen
- SLR- und LALR-Automat hat gleiche Zustandsmenge wie LR(0)-Automat
- Wie lautet die Übergangsfunktion?
- Wie bestimmt man Zustände und Übergänge des LALR(1)-Automaten von Hand?
- Wie korrigiert man Grammatik, wenn es inadäquate Zustände gibt?



## 3.3.1.1 Beispiele

- Abstrakte Beispiele:
  - <http://www.info.uni-karlsruhe.de/~heuzer/grammars.ps>
- 1. Grammatik:
  - Zerteiler-Generator: **antlr**
  - LL ( $k$ ) für  $k \geq 0$
  - Quellsprache: Java Spec. 2.0 für kjc/kaffe
  - Implementierungssprache: Java
- 2. Grammatik:
  - Zerteiler-Generator: **GNU bison**
  - LALR (1)
  - Quellsprache: Java für GNU gcj
  - Implementierungssprache: C



## Grammatik 1: antlr, Java

```
// Definition of a Java class
jClassDefinition[int mods]
returns [JClassDeclaration self = null]
{
    String                superClass = null;
    CClassType[]          interfaces = CClassType.EMPTY;
    CParseClassContext context      = new CParseClassContext ();
    TokenReference         sourceRef  = buildTokenReference ();
    JavadocComment        javadoc    = getJavadocComment ();
    JavaStyleComment[]    comments   = getStatementComment ();
}
:
"class" ident:IDENT
superClass = jSuperClassClause[] // it might have a superclass
interfaces = jImplementsClause[] // it might implement some interfaces
jClassBlock[context]             // the body of the class
{
    self = new JClassDeclaration(sourceRef,
                                mods, ident.getText (),
                                superClass, interfaces,
                                context.getFields (),
                                context.getMethods (),
                                context.getInnerClasses (),
                                context.getBody (),
                                javadoc, comments);
}
;
```



## Grammatik 2: bison, Java

```
/* 19.8.1 Production from §8.1: Class Declaration */
class_declaration:
modifiers CLASS_TK identifier super interfaces
{ create_class ($1, $3, $4, $5); }
class_body
{;}
| CLASS_TK identifier super interfaces
{ create_class (0, $2, $3, $4); }
class_body
{;}
| modifiers CLASS_TK error { yyerror ("Missing class name"); RECOVER; }
| CLASS_TK error { yyerror ("Missing class name"); RECOVER; }
| CLASS_TK identifier error
{
    if (!ctxp->class_err) yyerror ("{' expected");
    DRECOVER(class1);
}
| modifiers CLASS_TK identifier error
{ if (!ctxp->class_err) yyerror ("{' expected"); RECOVER; }
;
super:
{ $$ = NULL; }
| EXTENDS_TK class_type { $$ = $2; }
| EXTENDS_TK class_type error { yyerror ("{' expected"); ctxp->class_err=1; }
| EXTENDS_TK error
{ yyerror ("Missing super class name"); ctxp->class_err=1; }
;
```



## Kapitel 3: Zerteilung

1. Theoretische Grundlage: Kontextfreie Grammatiken
  - Notation
  - Konkrete und abstrakte Syntax
  - Rekursiver Abstieg
  - Kellerautomaten
  - Elimination von Linksrekursion und  $\epsilon$ -Produktionen
  - Systematische Zerteilerkonstruktion: LL-, LR-Grammatiken
2. LL- und SLL-Grammatiken
3. LR-, SLR-Grammatiken
  - 3.1 LALR-Konstruktion
  - 3.2 Optimierungen und Komplexität
4. Fehlerbehandlung



### 3.3.2 Tabellenoptimierung

LR(0)-Reduktionszustand: Zustand, in dem auf jeden Fall reduziert wird (Kontext unbeachtlich)

LR(0)-Reduktionszustände beseitigen, im vorigen Zustand bereits reduzieren (Schift-Reduktionszustand)

Kettenproduktionen eliminieren

echte und *unechte* (*don't care*) Fehlerübergänge unterscheiden.

Unecht: Übergang kann nie erreicht werden, z.B. alle leeren Übergänge mit Nichtterminalen

Fehlerübergänge ausfaktorisieren in Fehlermatrix  $F$ :

$$f(q,t) = \text{if } F[q,t] \text{ then Fehler else Eintrag\_in\_Übergangsmatrix}$$

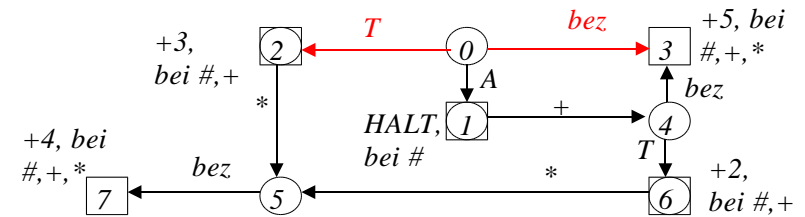
Übergangsmatrix komprimieren: leere Übergänge berücksichtigen

Übergangsmatrix weiter komprimieren (Graphfärbung, usw.)



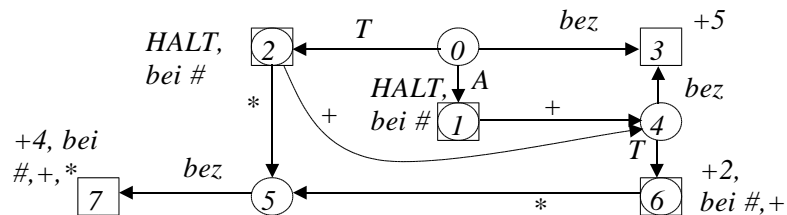
### 3.3.2 Kettenproduktionen

- (1)  $Z \rightarrow A$
- (2)  $A \rightarrow A + T$
- (3)  $A \rightarrow T$
- (4)  $T \rightarrow T^*bez$
- (5)  $T \rightarrow bez$



### 3.3.2 Kettenproduktionen Eliminieren

- (1)  $Z \rightarrow A$
- (2)  $A \rightarrow A + T$
- (3)  $A \rightarrow T$
- (4)  $T \rightarrow T^*bez$
- (5)  $T \rightarrow bez$



### 3.3.2 Tabelle mit Schift-Reduktionen, ohne Kettenproduktionen

	bez	(	)	+	*	#	A	T	F
0	-6	3	-	-	-	-	1	2	2
1	-	-	-	4	-	*	-	-	-
2	-	-	-	4	5	*	-	-	-
3	-6	3	-	-	-	-	6	7	7
4	-6	3	-	-	-	-	-	8	8
5	-6	3	-	-	-	-	-	-	-4
6	-	-	-7	4	-	-	-	-	-
7	-	-	-7	4	5	-	-	-	-
8	-	-	+2	+2	5	+2	-	-	-



### 3.3.2 Fehlermatrix und komprimierte Übergangsmatrix

		bez	(	)	+	*	#												
<table border="1"> <tr> <td>0,1,2</td> <td>1</td> <td>2</td> </tr> <tr> <td>3</td> <td>6</td> <td>7</td> </tr> <tr> <td>4</td> <td></td> <td>8</td> </tr> <tr> <td>5,6,7,8</td> <td></td> <td>-4</td> </tr> </table>	0,1,2	1	2	3	6	7	4		8	5,6,7,8		-4	0	f	f	t	t	t	t
	0,1,2	1	2																
	3	6	7																
	4		8																
	5,6,7,8		-4																
	1			t	f		f												
	2	t	t	t	f	f	f												
	3	f	f	t	t	t	t												
	4	f	f	t	t	t	t												
5	f	f	t	t	t	t													
6			f	f		t													
7	t	t	f	f	f	t													
8	t	t	f	f	f	f													

		bez	(	)	+	*	#
0,1,2,3,	-6	3	-7	4	5	*	
4,5,6,7							
8		+2	+2	5	+2		



### 3.3.2 Tabellengröße

Tabellengröße nach Elimination von Ketten (Beispiel ADA 83):

- 95 Terminale, 252 Nichtterminale
- 540 Zustände (kodierte in 2 Bytes)
- Tabellengröße: 374.760 Bytes

Reduktion des Platzbedarfes durch:

- Einfache Tabellenkomprimierung
- Elimination der Fehlerübergänge bei Nichtterminalen

Tabellengröße nach diesen Reduktionen 22.584 Bytes (ca. 6%)

Quelle: J. Grosch: "Lalr - A Generator for Efficient Parsers." GMD-Bericht 1988.



### 3.3.2 Vergleich verschiedener Zerteilergeneratoren

	Bison	yacc	PGS	Lalr	Ell
Grammatik Spezifiziert in	LALR(1) BNF	LALR(1) BNF	LALR(1) EBNF	LALR(1) EBNF	LL(1) EBNF
Geschwindigkeit in [10 <sup>3</sup> Symbol / Sekunde]	8.93	15.94	17.32	34.94	54.64
Geschwindigkeit in [10 <sup>3</sup> Zeilen / Minute]	150	270	290	580	910
Tabellengröße in [bytes] (komprimiert)	7724	9968	9832	9620	-
Zerteilergröße in [bytes]	10900	12200	14140	16492	18048

Eingabe: Modula-2 Code.

Hardware: PCS Cadmus mit MC68020 Prozessor (16.7 MHz).

Quelle: J. Grosch: "Lalr - A Generator for Efficient Parsers." GMD-Bericht 1988.



### 3.3.2 Studienarbeit zu vergeben

Neuer Vergleich der Zerteilergeneratoren

- Bison
- yacc
- PGS
- Lalr
- Ell

Auf aktuellen PC Architekturen um aktuelle Zahlen zu erhalten

Kann auch als Übersetzerbaupraktikum angerechnet werden

Freiwillige vor!



## 3.2.2 Komplexität Zerteilung in $O(n)$

Für **alle** Klassen gilt: Zerteilen benötigt Aufwand  $O(n)$

Jeder Knoten, da kein Rücksetzen  $O(1)$

Beweis über Grad der Knoten und mögliche Höhe des Baumes  
(Kettenproduktionen und Epsilonproduktionen berücksichtigen)



## 3.3.2 Komplexität

Grammatiktyp	Zerteiler- generierung	Test der Grammatik
LL(1)	$O(n^2)$	$O(n^2)$
SLL(k)	$O(n^{k+1})$	$O(n^{k+1})$
LL(k)	$O(2^{n^k + (k+1)\log n})$	$O(n^{2k})$
SLR(1)	$O(2^{n+\log n})$	$O(n^2)$
SLR(k)	$O(2^{n+k\log n})$	$O(n^{k+2})$
LR(k)	$O(2^{n^{k+1} + k\log n})$	$O(n^{2(k+1)})$



## Sätze über kontextfreie Grammatiken

**Satz 1:** Für jede LR(k)-Grammatik  $G$  mit  $k > 1$  gibt es eine LR(1)-Grammatik  $G'$  mit  $L(G) = L(G')$ .

Beweis durch Rechtsfaktorisierung.

**Satz 2:** Jede LL(k)-Grammatik ist auch LR(k).

**Satz 3:** Es gibt LR(k)-Grammatiken, die für kein  $k' \in \mathbb{N}$  LL(k') sind.

**Satz 4:** Es ist entscheidbar, ob es für eine gegebene LR(k)-Grammatik  $G$  ein  $k'$  gibt, so daß  $G$  LL(k') ist.

**Satz 5:** Es ist unentscheidbar, ob für eine Sprache  $L$  eine Grammatik  $G$  existiert, so daß  $G$  LL(1) ist.

**Satz 6:** Es ist unentscheidbar, ob es für eine Sprache  $L$  eine Grammatik  $G$  gibt, so daß  $G$  LL(k) oder LR(k) ist.



## Kapitel 3: Zerteilung

1. Theoretische Grundlage: Kontextfreie Grammatiken
  - Notation
  - Konkrete und abstrakte Syntax
  - Rekursiver Abstieg
  - Kellerautomaten
  - Elimination von Linksrekursion und  $\epsilon$ -Produktionen
  - Systematische Zerteilerkonstruktion: LL-, LR-Grammatiken
2. LL- und SLL-Grammatiken
3. LR-, SLR-Grammatiken
  - 3.1 LALR-Konstruktion
  - 3.2 Optimierungen und Komplexität
4. Fehlerbehandlung



## 3.4 Fehlerbehandlung

**Ziel:** Analyse soll formal korrekte Ergebnisse liefern (oder wegen zu vieler Fehler abbrechen), um möglichst viele Fehler zu finden.

- Gefahr von Folgefehlern wegen unzutreffender Korrektur unvermeidbar
- Unterscheide **Fehler** (= *Ursache*) und **Fehlersymptom** (= *beobachtbare Wirkung*)

**Reaktionen:**

- **Bericht:** Immer notwendig
- **Reparatur**, wenn Fehlerursache feststellbar und korrigierbar
- **Wiederaufsetzen:** Internen Zustand konsistent machen und weitere Fehler suchen
- **Abbruch** bei zu vielen Fehlern oder bei zu hohem Ressourcenverbrauch (Tabellenüberlauf, ...)



## 3.4 Fehlerbehandlung II

**Fehler:** Nicht (vom Programmierer) intendiertes Programm

**Fehlersymptom:**

- Sichtbare Auswirkung des Fehlers: Verletzung der Sprachdefinition
- zerteilerdefinierte Fehlerstelle.

**Diagnose:**

- Versuch, den Fehler auf der Grundlage des Symptoms zu erkennen.
- Entsprechende Fehlermeldung, Reparatur oder Wiederaufsetzen.



## 3.4 Beispiel

Zuweisung: `x := (a+b*c;`

Fehler (vermutlich): `)` nach `b` fehlt.

Fehlersymptom: `)` fehlt vor `;`

Position des Fehlersymptoms ist nicht die Fehlerstelle!



## 3.4 Klassifikation

**Fehlerklassen:**

- **Anomalien** (verdächtig, aber nicht gefährlich)
  - **Notiz** keine Standardkonstruktion
  - **Kommentar** unpassender Programmierstil
  - **Warnung** möglicher Fehler, z.B. unbenutzte Variable
- **Fehler**
  - **einfacher Fehler** reparierbar, Code kann erzeugt werden
  - **fataler Fehler** kein Code erzeugbar, nur Wiederaufsetzen möglich
  - **Abbruchfehler** Übersetzer gibt auf (z.B. wegen Ressourcenbeschränkung)



## 3.4 Fehlermeldung

- Ausgabe von
  - *Position* (Datei, Zeile, Spalte),
  - *Fehlerklasse*,
  - *Meldung*
- Intern: Meldung kodiert durch ganze Zahl
- Meldungstexte in getrennter Datei zur Anpassung der Sprache
- Meldungen fallen nicht in der Reihenfolge des Eingabetexts an:
  - Meldungen aufsammeln
  - nach Position sortiert ausgeben



## 3.4 Unterscheidung nach Übersetzerphase

1. **Symbolfehler**: unzulässiges Eingabezeichen, Abschluß Textkonstante oder Kommentar fehlt (falls erkennbar), verfrühtes Eingabeende: kein Endzustand des Symbolentschlüsselungsautomaten erreicht.  
**Reparatur**: falsche Zeichen oder vorhandenen Symbolanfang ignorieren
2. **Syntaktischer Fehler**: Satz gehört nicht zu der durch den Zerteiler definierten Obermenge der Sprache.
3. **Semantischer Fehler**: Fehler in der statischen Semantik.
4. **Semantischer Fehler**, der erst in der Optimierung entdeckt wird, z.B. Verletzung Indexgrenzen bei Reihungsindizierung.
5. **Ressourcenbeschränkung** verletzt (stets Abbruch), in allen Phasen möglich.



## 3.4 Zerteilerdefinierte Fehlerstelle

zerteilerdefinierte Fehlerstelle  $t$ :

$$\forall \underline{s} \in \Sigma^* : st\underline{s} \notin L \text{ und } \exists s' : ss' \in L$$

$LL$ -,  $SLL$ -,  $LR$ -,  $SLR$ - und  $LALR$ -Zerteiler finden die zerteilerdefinierte Fehlerstelle.

Andere Zerteiler, z.B für Präzedenzgrammatiken, finden sie nicht.



## 3.4 Beweisskizze für $LL(k)$

Angenommen  $t$  zerteilerdefinierte Fehlerstelle, d.h.  $st\underline{s} \notin L$ ,

$\exists s' : ss' \in L$ : Sei  $s$  im Keller und

$$q = [P \rightarrow P_{anf} \bullet P_{rest}; \Omega]$$

*Terminal*  $t$  ist zerteilerdefinierte Fehlerstelle gdw:

$$\neg \exists k : t\underline{s} \in \text{Anf}_k(P_{rest}; \Omega)$$



## 3.4 Problem am Beispiel

Grammatik:

- (1)  $Z \rightarrow I | A$
- (2)  $I \rightarrow \text{if } E \text{ then } Z \text{ end}$
- (3)  $A \rightarrow E := E$
- (4)  $E \rightarrow id E_{rest}$
- (5)  $E_{rest} \rightarrow e | = id$

SLL-Zerteilung des fehlerhaften Satzes:

**if**  $a := b$  **then** ... **end**

mit zerteilerdefiniertem Fehler  $:=$  landet in Situation

$[I \rightarrow \text{if } E \bullet \text{ then } Z \text{ end}; \Omega] := b \text{ then } \dots \text{ end}$

Beachte:  $:= \in \text{Folge}(E)$



## 3.4 Problem am Beispiel

leicht modifizierte Grammatik:

- (1)  $Z \rightarrow I | A$
- (2)  $I \rightarrow \text{if } B \text{ then } Z \text{ end}$
- (3)  $A \rightarrow E := E$
- (4)  $B \rightarrow E$
- (5)  $E \rightarrow id E_{rest}$
- (6)  $E_{rest} \rightarrow e | = id$

Zerteilung von: **if**  $a := b$  **then** ... **end**

mit zerteilerdefiniertem Fehler  $:=$  landet in Situation

$[I \rightarrow \text{if } E \bullet \text{ then } Z \text{ end}; \Omega] := b \text{ then } \dots \text{ end}$

Beachte:  $:= \notin \text{Folge}(B)$



## 3.4 Ideale Korrektur

Minimale Anzahl von Operationen

- Einsetzen
- Streichen
- Ersetzen (= Streichen + Einsetzen)

ausführen



## 3.4 Tatsächliche Korrektur

- Panischer Modus: Wiederaufsetzen an Anweisungs- oder Vereinbarungsende
- Systematische Fortsetzung an zerteilerdefinierter Fehlerstelle  $st_{\underline{s}} \notin L$ : frühest möglichen Wiederaufsetzpunkt finden
- Totalkorrektur:
  - Eingabe  $st_{\underline{s}} = s_1 x_1 \dots s_n x_n$ ,  $s_1 = s$ ,  $n$  minimal, so aufteilen, daß alle  $s_i$  Ausschnitte einer korrekten Eingaben sind, die durch eventuell unbrauchbare Texte  $x_i$  verknüpft sind
  - die  $x_i$  durch korrekte Texte  $y_i$  so ersetzen, daß  $s_1 y_1 \dots s_n y_n$  korrekt ist
  - **Nachteil**: quadratischer Aufwand, praktisch nicht eingesetzt

zahlreiche weitere Verfahren bekannt





## 3.4 Panischer Modus

alle Symbole bis Anweisungs- oder Vereinbarungsende streichen  
Keller soweit abbauen, daß Folgesymbol ; end, }, ... akzeptiert wird  
Fehlermeldung ausgeben und Analyse fortsetzen

### Vorteil:

- einfache Implementierung

### Nachteile:

- keine Analyse des Anweisungsrests
- Schwierigkeiten mit korrektem Abschluß von Klammerungen  
**if ... then, then ... else, usw.**



## 3.4 C (gcc) - Panischer Modus

```
int main ( ) {
  int j, i ;
  if (i<j) { {           /*FEHLER*/
    I = j ;             /*FEHLER-SYMP TOM 1: } erwartet*/
  }
  else {
    if ( i != j ){
      j:= i ;           /*FEHLER 2 bleibt unerkant*/
    }                  /*Ende if */
  }                  /*Ende main*/
  return 0 ;          /*FEHLER-SYMP TOM 2: return unerwartet*/
}
-----
test.c: In function ´main´:
test.c: 6: parse error before ´else´
test.c: At top level:
test.c: 11: parse error before ´return´
```



## 3.4 Sather-K (fiasco) - Ad Hoc

```
class TEST is
main is
  j,i: INT;
  if i < j then then      /* FEHLER 1 */
    i := j ;
  else
    if i /= j then
      j := i_           /* FEHLER 2 bleibt unerkant */
    end;
  end;
end;
end;
end;
-----
test.sa:4: parse error, expecting ´TOK_END´
if i < j then then      -- FEHLER 1
```



## 3.4 Systematische Korrektur (Röhrich)

gegeben zerteilerdefinierter Fehlerstelle  $str \notin L$

gesucht Fortsetzung sy in L:

- Bestimme eine **Ankermenge**  $D = \{ d \in \Sigma / y = ss'ds'' \in L \}$
- Suche ein  $d \in D$ , so dass  $r = r'dr''$  und  $|r'|$  minimal ist.
- Ersetze  $tr'dr''$  durch  $s'dr''$ .
- gib Fehlermeldung aus und setze Analyse fort

$$\frac{s tr = s tr'dr''}{s s' dr''}$$

### Vorteile:

- nahe bei Minimal Korrektur, einfache Fehlermeldung
- fast vollautomatisch erzeugbar
- terminiert, da Eingabe um  $d$  verkürzt.

### Nachteile:

- Korrektur nicht zwangsläufig korrekt
- Schwierigkeiten bei Listenkonstruktionen (Anweisungs-, Bezeichner-, Vereinbarungslisten, usw.)
- bei rekursivem Abstieg Vorbereitung notwendig
- bei LR-Analyse Adaption des Generators notwendig



## 3.4 ADA (adac) - Fehlerbehandlung

```
procedure Main is
  I;J : Integer;
begin
  if I > J then then /* FEHLER 1: I in Ankermenge,
                    -   then gestrichen */
    I:=J
  else
    if I /= J then
      J = ; I /* FEHLER 2: = unerwartet,
              ; in Ankermenge, = gestrichen */
    end if /* SYMPTOM: ; erwartet, end in
            Ankermenge,
            ; eingefügt */
  end if;
end Main;
```



## 3.4 ADA (adac) - Fehlerausgabe

```
error.ada...
4, 19: Error syntax error
4, 19: Information expected tokens:
IDENTIFIER CHAR_STRING NULL PRAGMA CASE RETURN
FOR BEGIN EXIT GOTO DELAY ABORT RAISE REQUEUE IF
WHILE LOOP DECLARE ACCEPT SELECT <<
5, 3: Information restart point
8, 7: Error syntax error
8, 7: Information expected tokens: . ; ( , : : =
8, 9: Information restart point
8, 6: Error syntax error
9, 6: Information expected tokens: . ; ( , : : =
9, 6: Repair token inserted : ;
```



## 3.4 Durchführung

1. Zeichne in jedem Zustand eine Situation aus, deren weitere Verfolgung zur Generierung der Fortsetzung führt. Die Fortsetzung muß terminieren. Ankermenge sind alle Symbole in dieser Fortsetzung.
2. Streiche in der Eingabe alle Zeichen bis zum ersten Zeichen in der Ankermenge.
3. Dieses Zeichen wird während der Verarbeitung der generierten Fortsetzung in einem Zustand  $q'$  akzeptiert und es gibt einen Übergang vom Fehlerzustand  $q$  nach  $q'$
4. Setze die Zeichen ein, die den Automaten vom Zustand  $q$  nach  $q'$  bringen.



## 3.4 LL Durchführung

- Zeichne für jedes Nichtterminal eine Produktion aus, die in der Fortsetzungserzeugung vorhergesagt wird. Die Produktion darf nicht rekursiv sein!
- Bestimme für die laufenden und jede vorhergesagte Produktion die noch fehlenden Symbole. Diese bilden die Ankermenge. Die noch laufenden Produktionen sind aus dem Keller ersichtlich.
- Schwierigkeit bei rekursivem Abstieg: Keller nur nach Prozedurrückkehr zugänglich.
  - Abhilfe: Ankermenge bereits während der normalen Zerteilung aufbauen und in getrenntem Keller (Datenstruktur) ablegen oder als Argument bei Prozeduraufruf mitgeben.



### 3.4 LL Beispiel Automat

$Z \rightarrow A$	$*q_0 i \rightarrow q_1 q_2 i, q_0 (\rightarrow q_1 q_2 (,$
$A \rightarrow FA'$	$*q_1 \rightarrow \epsilon,$
$A' \rightarrow \epsilon   + FA'$	$*q_2 i \rightarrow q_3 q_4 i, q_2 (\rightarrow q_3 q_5 (,$
$F \rightarrow i   (A)$	$*q_3 \# \rightarrow q_6 q_7 \#, q_3 (\rightarrow q_6 q_7), q_3 \rightarrow q_6 q_8 +,$
$q_0 : [Z \rightarrow \bullet A]$	$q_8 : [A' \rightarrow \bullet + FA']$
$q_1 : [Z \rightarrow A \bullet]$	$q_9 : [F \rightarrow i \bullet]$
$q_2 : [Z \rightarrow \bullet FA]$	$q_{10} [F \rightarrow (\bullet A)]$
$q_3 : [A \rightarrow F \bullet A]$	$q_{11} [A' \rightarrow + \bullet FA']$
$q_4 : [F \rightarrow \bullet i]$	$q_{12} [F \rightarrow (A \bullet)]$
$q_5 : [F \rightarrow \bullet (A)]$	$q_{13} [A' \rightarrow + F \bullet A']$
$q_6 : [A \rightarrow FA \bullet]$	$q_{14} [F \rightarrow (A) \bullet]$
$q_7 : [A' \rightarrow \bullet \epsilon]$	$q_{15} [A' \rightarrow + FA \bullet]$

$*q_4 i \rightarrow q_9,$   
 $*q_5 (\rightarrow q_{10},$   
 $*q_6 \rightarrow \epsilon,$   
 $*q_7 \rightarrow \epsilon,$   
 $*q_8 + \rightarrow q_{11},$   
 $*q_9 \rightarrow \epsilon,$   
 $*q_{10} i \rightarrow q_{12} q_2 i, q_{10} (\rightarrow q_{12} q_2 (,$   
 $*q_{11} i \rightarrow q_{13} q_4 i, q_{11} (\rightarrow q_{13} q_5 (, *q_{12} ) \rightarrow q_{14},$   
 $*q_{13} \# \rightarrow q_{15} q_7 \#, q_{13} (\rightarrow q_{15} q_7), q_{13} + \rightarrow q_{15} q_{15} +,$   
 $*q_{14} \rightarrow \epsilon, *q_{15} \rightarrow \epsilon.$

**\* zeichnet die vorherzusagenden Produktionen bzw. Zustandsübergänge aus**

### 3.4 LL Beispiel i + #

Zerteilen bis zum Fehler	Fortsetzung finden	Wiederaufsetzen
$q_0 i + \#$	$q_1 q_6 q_{11}$	$D = \{ i ( \}$ $q_1 q_6 q_{11} \#$
$q_1 q_2 i + \#$	$q_1 q_6 q_{13} q_4$	$q_1 q_6 q_{13} q_4 \#$
$q_1 q_3 q_4 i + \#$	$q_1 q_6 q_{13} q_9$	$q_1 q_6 q_{13} q_9 \#$ <i>i generiert mit</i>
$q_1 q_3 q_9 + \#$	$q_1 q_6 q_{13}$	$D = \{ i ( \# ) + \}$ $q_4 i \rightarrow q_9$
$q_1 q_3 + \#$	$q_1 q_6 q_{15} q_7$	$q_1 q_6 q_{13} \#$ <i>weiter normal</i>
$q_1 q_6 q_8 + \#$	$q_1 q_6 q_{15}$	
$q_1 q_6 q_{11} \#$	$q_1 q_6$	
	$q_1$	

### 3.4 LR Durchführung

- Zeichne für jedes Nichtterminal eine nichtrekursive Produktion aus
- Zustände sind nicht Mengen, sondern geordnete Listen von Situationen. Die Situationen für ausgezeichnete Produktionen kommen jeweils vor allen anderen Situationen mit gleicher linker Seite.
- Erzeugung der Fortsetzung: Benutze in jedem Zustand jeweils die erste Situation und ihren Übergang.
- Die ausgezeichnete Situation gehört zur Basis des Zustandes.
- Verfahren terminiert wegen Nichtrekursivität der ausgezeichneten Produktionen
- Sonderbehandlung von Trennzeichen für Listen. Sie werden initial in entsprechende Ankermenge übernommen.

### 3.4 LR Beispiel-Automat

0: $Z \rightarrow \bullet A ; \#$	4: $F \rightarrow (\bullet A) ; \# +$
$A \rightarrow \bullet F ; \# +$	$A \rightarrow \bullet F ; ) +$
$A \rightarrow \bullet A + F ; \# +$	$A \rightarrow \bullet A + F ; ) +$
$F \rightarrow \bullet bez ; \# +$	$F \rightarrow \bullet bez ; ) +$
$F \rightarrow \bullet (A) ; \# +$	$F \rightarrow \bullet (A) ; ) +$
(1) $Z \rightarrow A \#$	
(2) $A \rightarrow A + \bullet ; \#$	5: $A \rightarrow A + \bullet F ; \# +$
(3) $A \rightarrow F$	$F \rightarrow \bullet bez ; \# +$
(4) $F \rightarrow bez$	$F \rightarrow \bullet (A) ; \# +$
(5) $F \rightarrow (A)$	6: $F \rightarrow (A \bullet) ; \# +$
	$A \rightarrow A \bullet + F ; ) +$
	7: $A \rightarrow A + F \bullet ; \# +$
	8: $F \rightarrow (A) \bullet ; \# +$

## 3.4 Übergänge

q	bez	(	)	+	#	A	F
0	-4	4	-	-	-	1	-3
1	-	-	-	5	*1		
4	-4	4	-	-	-	6	-3
5	-4	4	-	-	-		-2
6	-	-	-5	5	-		

Grau hinterlegt: Ausgezeichnete Übergänge



## 3.4 LR Beispiel $i + )i \#$

Zerteilen bis Fortsetzung finden Wiederaufnahme  
zum Fehler

$q_0 i + ) i \#$        $q_0 q_1 q_5$      $D = \{ i ( \}$        $q_0 q_1 q_5 i \#$  wobei  $)$  gelöscht  
 $q_0 q_1 + ) i \#$        $q_0 q_1$        $D = \{ i ( + \# \}$       weiter normal  
 $q_0 q_1 q_5 ) i \#$



## 3.4 Fehlerbehandlung in der semantischen Analyse

für jede Eigenschaft oder Wertetyp einen ausgezeichneten Wert *unbekannt* definieren

- Operationen mit *unbekannt* liefern wieder *unbekannt*
- Fehlermeldung nur, wenn *unbekannt* als Ergebnis, nicht als Operand erscheint.

Quellen fataler Fehler:

- fehlende Vereinbarung: Bezeichner erhält Typ *unbekannt* (oder Typ durch Typinferenz aus Anwendung erschließen)
- unzulässige Operation: Ergebnistyp ist *unbekannt*
- inkompatible Operanden oder sonstige Verletzung von Konsistenzbedingungen: Fehler melden, Ergebnistyp *unbekannt*, mit Analyse fortfahren

