3. Kapitel

syntaktische Analyse

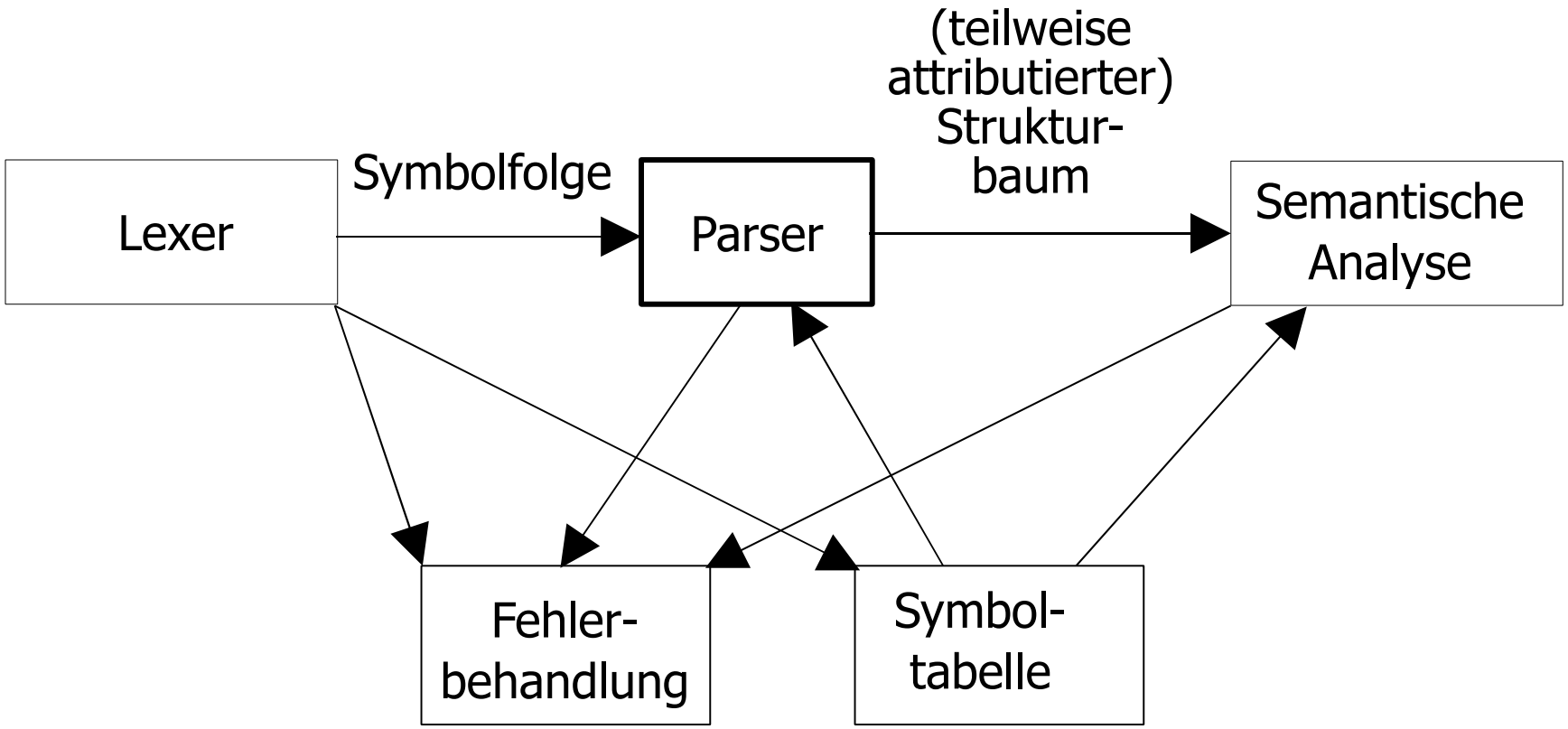
Kapitel 3: syntaktische Analyse

- 0. Einbettung
- 1. Theoretische Grundlage: Kontextfreie Grammatiken
  Notation
  Konkrete und abstrakte Syntax
  Kellerautomaten
  Elimination von Linksrekursion und ε-Produktionen
  Systematische Parserkonstruktion: LL-, LR-Grammatiken
- 2. LL- und SLL-Grammatiken
- 3. LR-, SLR-Grammatiken
  3.1 LALR-Konstruktion
  3.2 Optimierungen und Komplexität
- 4. Fehlerbehandlung

3. Syntaktische Analyse

- vorgegeben:
  - Tokenstrom
  - kontextfreie Grammatik (deterministisch?)
- Aufgaben
  - syntaktische Struktur bestimmen
  - syntaktische Fehler melden, korrigieren (?)
  - Ausgabe (immer): abstrakte Syntax (Rechts-/Linksableitung), Symbole (Bezeichner, Konstanten, usw.)

3. Einbettung des Parsers



3. ADT Parser

- gelieferte Operationen:

```
initialize
quit
parse
```

- benötigte Operationen:
  - Lexer/Tokenstrom:
    `next_token() : Token`
  - Fehlerbehandlung
    `add_error(nr, pos)`
  - für Aufbau Strukturbaum
    `production(nr), symbol(value)`

3. Aufgabe des Parsers, formal

Gegeben: Grammatik $G=(T,N,P,Z)$ mit
T Alphabet, N Nichtterminale, P Produktionen, Z Zielsymbol

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

wenn ja, Produktionenfolge für Links-/Rechtsableitung
wenn nein, Fehlerbehandlung zur Korrektur des Tokenstroms.

Unterscheide konkrete Syntax G_k und abstrakte Syntax G_a:

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

Beziehung zwischen G_k und G_a?

3.1 Annahmen für das Parsen

Syntax ist kontextfrei

- eigentlich ist sie kontext-sensitiv
- aber kontext-sensitive Grammatiken nicht in linearer Zeit parsbar (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 lexikalischen Analyse

- sonst gäbe es mehrere Grundzustände des Lexers, gesteuert vom Parser

keine Rückkopplung semantische Analyse – syntaktische Analyse

- typunabhängige Syntaktische Analyse: Zustände des Parsers unabhängig von der Namens- und Typanalyse

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 kfG-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: Syntaktische Analyse

- 1. Theoretische Grundlage: Kontextfreie Grammatiken
  Notation
  Konkrete und abstrakte Syntax
  Kellerautomaten
  Elimination von Linksrekursion und ε-Produktionen
  Systematische Parserkonstruktion: 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: $A \rightarrow x \mid y \mid \ldots, A \in N, x,y \in V^*, V = T \cup N$

praktisch: Backus-Naur-Form (BNF)

- Nichtterminale in spitzen Klammern,
- Terminale als Symbole oder wie Nichtterminale
- ::= statt →

<Ausdruck> ::= <Term> | <Ausdruck> + <Term>

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

Ausdruck ::= Term ('+' Term)* .

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

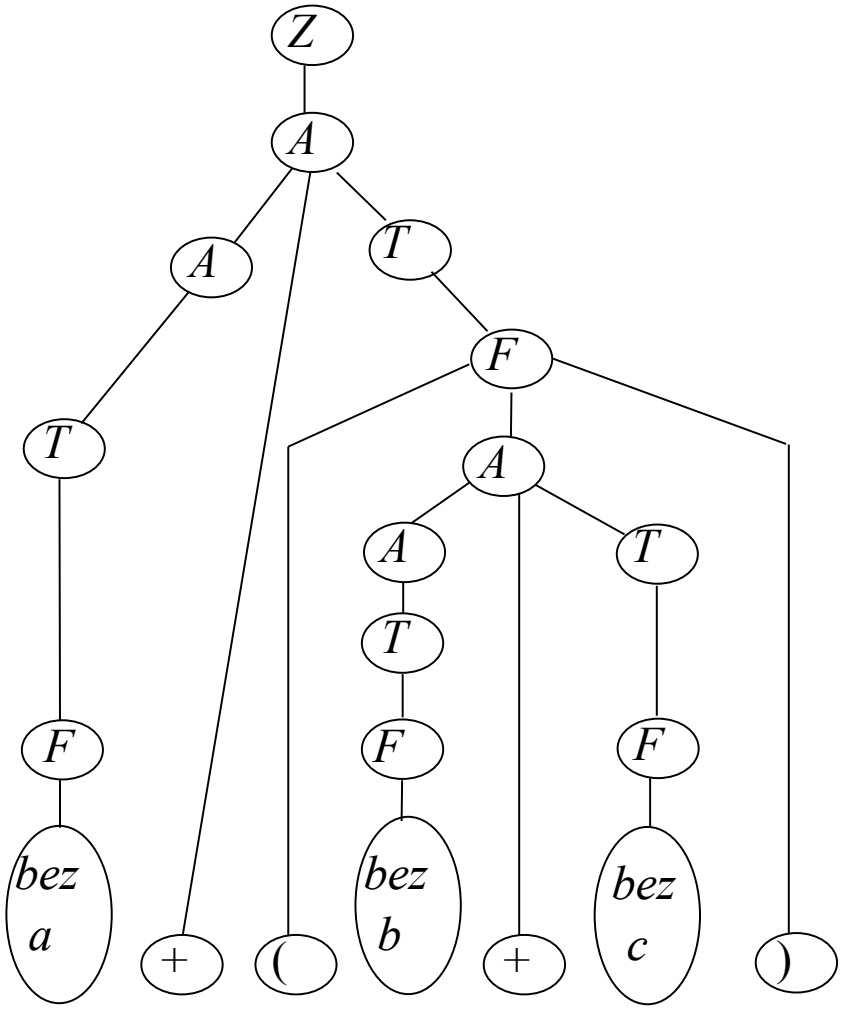
3.1 EBNF mit Listennotation

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

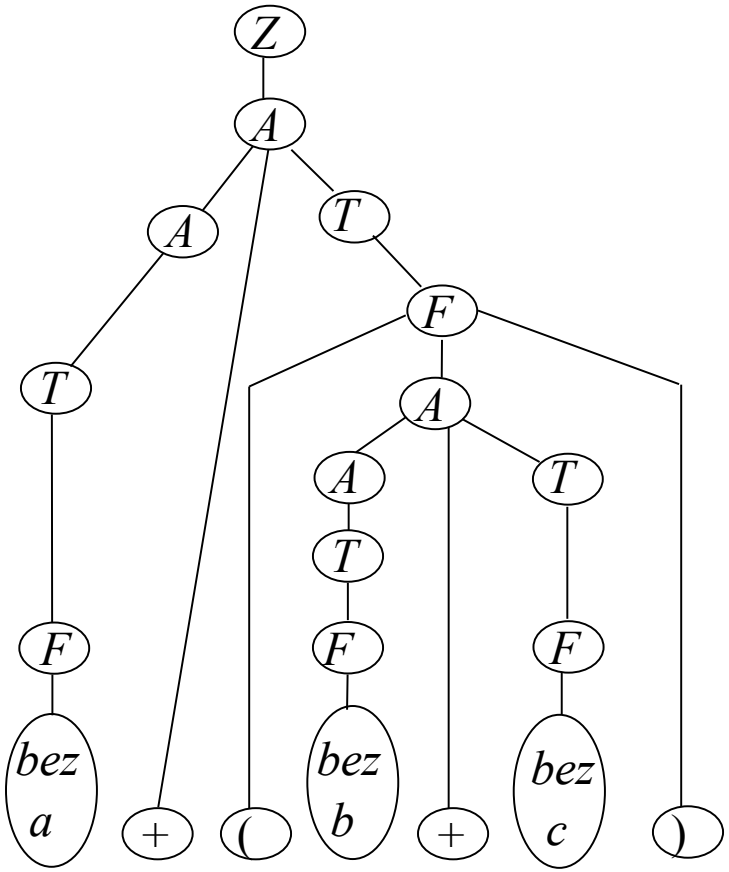
3.1 Beispielgrammatik

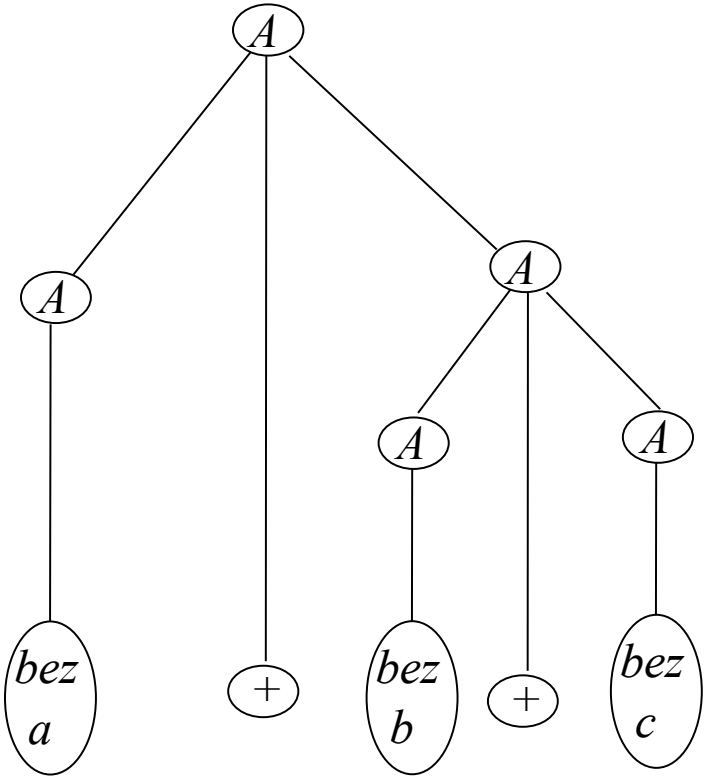
- Ausdrücke
  (0) $Z \rightarrow A$
  (1) $A \rightarrow T$ (2) $A \rightarrow A + T$
  (3) $T \rightarrow F$ (4) $T \rightarrow T * F$
  (5) $F \rightarrow bez$ (6) $F \rightarrow (A)$
- EBNF:
  (0) $Z \rightarrow A$.
  (1) $A \rightarrow T$ ($'+'$ T)* .
  (2) $T \rightarrow F$ ($'*'$ F)* .
  (3) $F \rightarrow bez \mid '('\ A\ ')'$.
- Ausdruck *a + (b+c)*
- konkreter Strukturbaum:



3.1 Konkrete und abstrakte Syntax

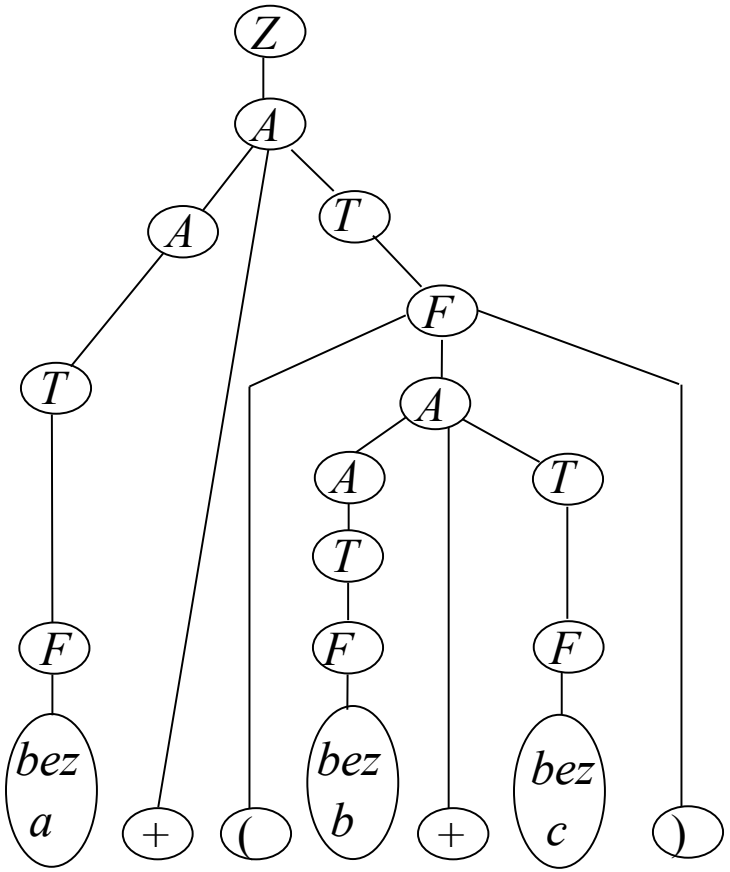
- Prinzip der abstrakten Syntax: nur die für die Semantik wichtige Struktur behalten: $A \rightarrow A + A \mid A * A \mid bez$

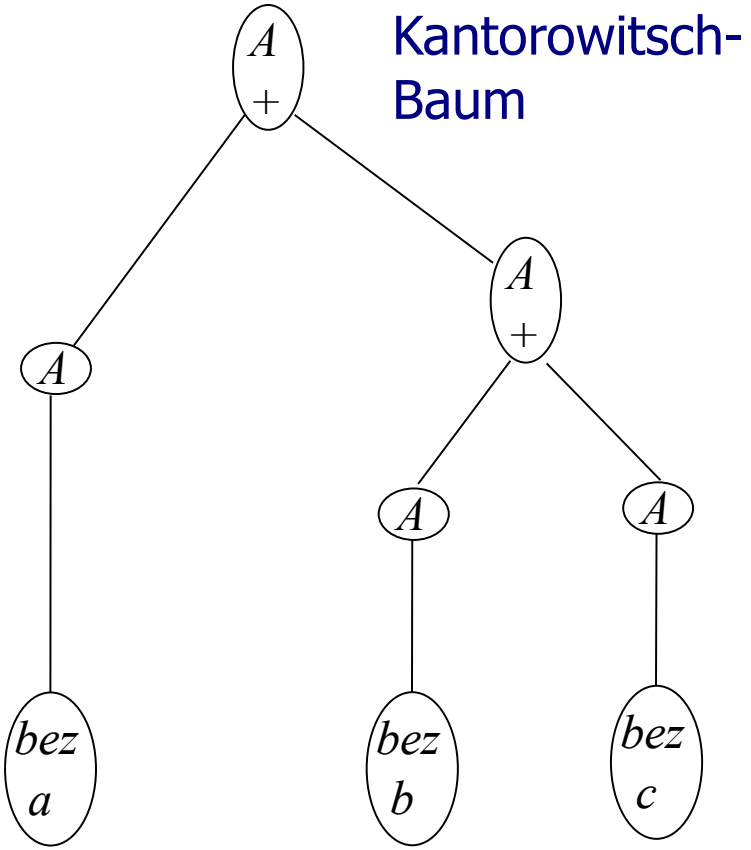




3.1 Konkrete und abstrakte Syntax

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





Kantorowitsch-Baum

3.1 Übergang: Konkrete und abstrakte Syntax

Konkrete Syntax G_k der zu übersetzenden Sprache (Datenstruktur: Tokenstrom)

1. explizite Strukturinformation `( ), begin end`, etc.
2. Ketten- und Verteilerproduktionen $A \rightarrow B$ bzw. $A \rightarrow B \mid C \mid \ldots$
3. Schlüsselwörter

Abstrakte Syntax G_a des Strukturbaums (Datenstruktur: Baum, AST).

1. Klammerung durch AST bereits eindeutig
2. Kettenproduktionen überflüssig, wenn keine semantische Bedeutung
3. Schlüsselwörter dienten der eindeutigen Zerteilung, jetzt immer überflüssig, werden weggelassen.

- Abbildung konkrete auf abstrakte Syntax durch Parser (Verarbeitung von Anknüpfungen), ggf. weitere Transformation während semantischer Analyse
- Produktionsnummer wird Knotentyp
- Operatoren als Attribute des Knotens für den Ausdruck

3.1 Sonderfälle in abstrakter Syntax

- Bezeichner:
  - $A \rightarrow bez$ ist Kettenproduktion,
    soll aber wegen semantischer Analyse erhalten bleiben
- Klammern in Fortran:
  - Information eigentlich bereits in der Baumstruktur
  - aber Klammern sind bindend (kein Umordnen erlaubt)
  - sonst gilt eventuell Assoziativgesetz (Umordnen möglicherweise 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 Parsers: Produktionen von G_a und Symbole

Methode: Anknüpfung der Ausgaberoutinen in die Grammatik eintragen

Syntaktische Anknüpfungen: %Ausgabe

- Nach Erkennen des vorgehenden (Nicht-)Terminals ausgeführt.
- Für AST: Konstruktor des entsprechenden Knotens im Ableitungsbaum für G_a aufrufen.

Symbolanknüpfungen: &Ausgabe

- Wird ausgeführt, wenn Symbol erkannt aber noch nicht fortgeschaltet wurde.
- Für AST: Konstruktoren werden gegebenenfalls Merkmale von Symbolen übergeben.

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

3.1 Beseitigung von ϵ-Produktionen

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

Technik dazu: ϵ-Abschluß

Einsetzen von Ableitungen der Form $A \rightarrow \epsilon$

in alle rechten Seiten der Form $X \rightarrow \alpha A \beta$, mit $\alpha, \beta \in (T \cup N)^*$

3.1 Linksfaktorisierung

Produktionen $X \rightarrow Yb \mid Yc$ 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 $X \rightarrow Yb \mid Yc$ durch $X \rightarrow YX', X' \rightarrow b \mid c$

Analog kann man bei LR-Analyse rechtsfaktorisieren (seltener benötigt).

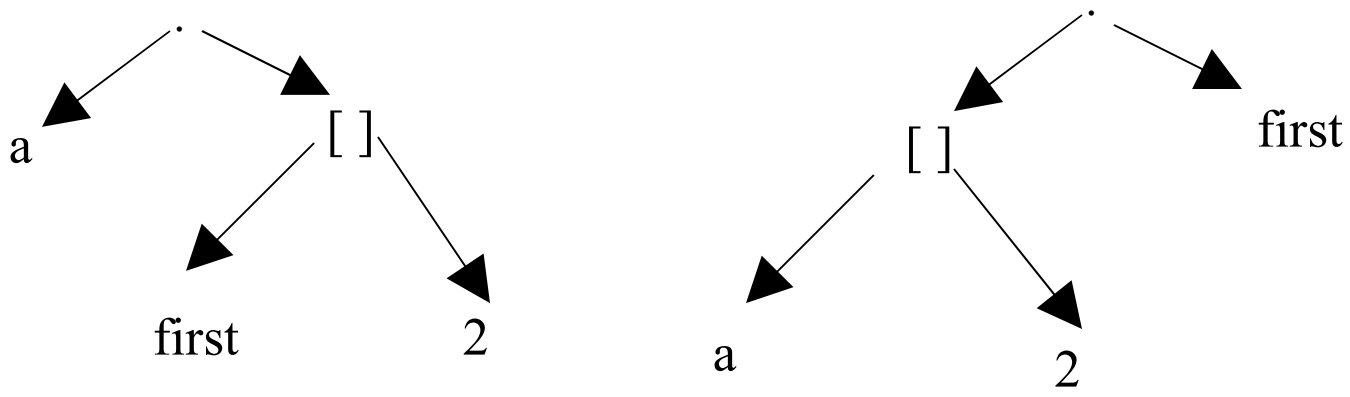
3.1 Typunabhängiges Parsen

Typunabhängiges Parsen

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

Typabhängiges Parsen

- Bsp: ADA a.first(2)



3.1 Typabhängiges Parsen

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

Parser umschaltbar, um Formate zu bearbeiten

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

Umschaltung also semantik- (oder typ-) gesteuert

Ähnliche Probleme in ABAP/4

3.1 Systematische Parserkonstruktion

- Es gibt weit mehr als 25 verschiedene Techniken zur Parserkonstruktion, vgl. Aho&Ullman, The Theory of Parsing and Compiling, 2 Bde, 1972
- Nur zwei Techniken , LL und LR, haben die Eigenschaften:
  - Der Parser liest die Quelle einmal von links nach rechts und baut dabei die Links- bzw. Rechtsableitung auf (daher die 2 Buchstaben).
  - Der Parser erkennt einen Fehler beim ersten Zeichen t, das nicht zu einem Satz der Sprache gehören kann. t heißt parserdefinierte Fehlerstelle (parser defined error):
    Wenn $x \notin L(G)$ und der Parser 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 Ausgabe von Postfix- oder Präfixform

Ausgaberoutinen:

addop: gib aus '+'
mulop: gib aus '*'
bezeichner: gib aus *bez*
merke: merke *bez*
bez_aus: gib gemerkten *bez* aus, falls vorhanden

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

3.1 Kellerautomaten

- $A = (T,Q,R,q_0,F,S,s_0)$

  T Eingabealphabet (Tokens)

  Q Zustandsmenge

  R Menge von Regeln $sqx \rightarrow s'q'x'$, $s,s' \in S^*$, $q,q' \in Q$, $x \in T^*$, $x = x''x'$

  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_0q_0y, y vollständige Eingabe
- Regel $sqx \rightarrow s'q'x'$ anwendbar, wenn $\underline{s} = \underline{s'}\underline{s}$, $\underline{x} = x\underline{x'}$
- Folgekonfiguration: $\underline{s'}s'q'x'\underline{x'}$
- Halt bei Konfiguration sq, $q \in F$, Eingabe vollständig gelesen
  praktisch Endezeichen # erforderlich, Halt bei $sq\#$

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

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

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

Aber: Sprachinklusion und Gleichheit nicht entscheidbar

⇒ keine eindeutige Normalform

3.1 Textmengen

$k : x = \quad x\#, \qquad$ falls $\qquad x = x_1 \ldots x_m \wedge m < k$

$k : x = \quad x_1 \ldots x_k\,, \quad$ falls $\qquad x = x_1 \ldots x_m \wedge m \geq k$

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

in der Literatur auch $\text{First}_k(x)$ genannt

$\text{Anf}'_k(x) = \quad \{u \mid u \in \text{Anf}_k(x) \wedge \neg \exists A \in N,\ y \in T^* \;$ so daß $x \Rightarrow^R Auy \Rightarrow uy\}$

in der Literatur auch $\text{EFF}_k(x)$ genannt (*EFfective First*)

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

in der Literatur auch $\text{Follow}_k(x)$ genannt

3.1 Herleitung der LL- und LR-Parser

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

Für LL: (prädiktiv)	Für LR: (reduzierend)
$tqt \rightarrow q,\ t \in T$	$qt \rightarrow tq,\ t \in T$
$Xq \rightarrow x_n \ldots x_1 q,\ X \rightarrow x_1 \ldots x_n \in P$	$x_1 \ldots x_n q \rightarrow Xq,\ X \rightarrow x_1 \ldots x_n \in P$

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

3.1 Nichtdeterministische LL- und LR-Parser

•Für LL: (prädiktiv)
Vergleich (compare):

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

Vorhersage (produce):

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

top-down Parser

vom Startsymbol zum Wort

•Für LR: (reduzierend)
Schift (shift):

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

Reduktion (reduce):

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

bottom-up Parser

vom Wort zum Startsymbol

Anmerkung: Der Zustand q ist noch bedeutungslos, er wird später beim deterministisch Machen benötigt.

Kapitel 3: syntaktische Analyse

- 1. Theoretische Grundlage: Kontextfreie Grammatiken
  Notation
  Konkrete und abstrakte Syntax
  Kellerautomaten
  Elimination von Linksrekursion und ε-Produktionen
  Systematische Parserkonstruktion: 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 $G=(T,N,P,Z)$ eine $LL(k)$-Grammatik, wenn

für alle Paare von Ableitungen

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

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

gilt:

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

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 ϵ-Produktionen.

3.2 Beispiele von LL-Grammatiken

- $A \rightarrow TA',\ A' \rightarrow \epsilon \mid +TA',\ T \rightarrow FT',\ T' \rightarrow \epsilon \mid *FT',\ F \rightarrow bez \mid (\ A\ )$ ist $LL(1)$.
- $Z \rightarrow aAab \mid bAbb,\ A \rightarrow \epsilon \mid a$ ist $LL(2)$, nicht $LL(1)$: Vorschau aa, ab, bb entscheidet unter Berücksichtigung der Produktion für Z über die Produktion für A.
- $Z \rightarrow X,\ X \rightarrow Y \mid bYa,\ Y \rightarrow c \mid ca$ ist $LL(3)$.
- $Z \rightarrow X,\ X \rightarrow Yc \mid Yd,\ Y \rightarrow a \mid bY$ 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 vorhersagbar. Bei Beginn mit Bezeichner sind Umformungen nötig.

3.2 Satz über Linksrekursion

- Satz: Eine linksrekursive kfG ist für kein k $LL(k)$.

- 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 beliebig $X_1,...,X_n$.
  - Für $i=1,...,n$
    - Für $j=1,...,i-1$ ersetze $X_i \rightarrow X_j x$ durch $\{X_i \rightarrow y_j x \mid X_j \rightarrow y_j \in P\}$
      (danach $i \leq j$, wenn $X_i \rightarrow X_j x \in P$)
    - ersetze die Produktionenmengen $\{X_i \rightarrow X_i x\} \cup \{X_i \rightarrow z \mid z \neq X_i z'\}$ durch
      $\{Y_i \rightarrow x Y_i \mid X_i \rightarrow X_i x \in P\} \cup \{Y_i \rightarrow \mathrm{e}\} \cup \{X_i \rightarrow z Y_i \mid X_i \rightarrow z \in P \text{ und } z \neq X_i z'\}$
      mit einem neuen Nichtterminal Y_i.
      (Numerierung der Y_i mit $n+1, n+2,...$)
  - Ergebnis: $i<j$, wenn $X_i \rightarrow X_j x \in P$
- Beachte: in Schritt 2 Ersetzung durch
  $\{Y_i \rightarrow x,\ Y_i \rightarrow x Y_i \mid X_i \rightarrow X_i x \in P\} \cup \{X_i \rightarrow z,\ X_i \rightarrow z Y_i \mid X_i \rightarrow z \in P \text{ und } z \neq X_i z'\}$
  ohne ϵ-Produktionen möglich, wenn x nicht mit X_j, $j \leq i$, beginnt.

3.2 Beispiel

- $A \rightarrow T \mid A + T,\ T \rightarrow F \mid T * F,\ F \rightarrow bez \mid (A)$
  ist linksrekursiv
- Ersetzung: Schritt 1 leer, Schritt 2: $A \rightarrow T \mid A + T$ durch
  $A \rightarrow T\,A',\ A' \rightarrow e \mid + TA'$ ersetzen; $T \rightarrow F \mid T * F$ analog.
  Dies entspricht der EBNF
  $A \rightarrow T\ (\ '+'\ T)^*,\ T \rightarrow F\ (\ '*'\ F)^*,\ F \rightarrow bez \mid '('\ A\ ')'$.
- Andere mögliche Ersetzung $A \rightarrow T \mid T\,A',\ A' \rightarrow + T \mid TA'$
- Vorsicht!
  Die Ersetzung durch $A \rightarrow T \mid T + A$ 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 <*Bezeichner*> <*Operator*> anfangen können (Zuweisungen, Ausdrücke)

3.2 $SLL(k)$-Grammatiken

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

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

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 $SLL(\mathrm{k})$-Grammatik, wenn für alle Paare von Produktionen $A \to x \mid x',\ x \neq x'$, die $SLL(k)$-Bedingung gilt:

$$\mathrm{Anf}_k(x\ \mathrm{Folge}_k(A)) \cap \mathrm{Anf}_k(x'\ \mathrm{Folge}_k(A)) = \varnothing$$

- Beweis: trivial
  - Also: $SLL(k)$-Eigenschaft durch Berechnung von Anf_k– und Folge_k–Mengen einfach nachzuprüfen.
  - Wenn aus x, x' nur terminale Zeichenreihen mit mindestens k Zeichen ableitbar sind, trägt $\mathrm{Folge}_k(A)$ nichts zum Ergebnis bei und kann entfallen.
  - Anwendung: Für $k = 1$ entfällt $\mathrm{Folge}_1(A)$ bei ε–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 $A \to x \mid x',\ x \neq x',$ 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:
  $Z \to aAab \mid bAbb,\ A \to \varepsilon \mid a$ 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 $X \rightarrow x_1 \ldots x_n$
- Übergänge $tqt \rightarrow q$ (bei LL) bzw. $qt \rightarrow tq$ (bei LR) sind nur zulässig, wenn in der Produktion ein Terminalzeichen t ansteht, $x_1 \ldots x_n = x'tx''$, wobei $x'' := tx'''$
- also: ersetze sqx durch Situation $[X \rightarrow x' \bullet x'' ; x]$, die durch den Punkt anzeigt, wieweit 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.
- Situationen heißen englisch *items*.

3.2 Formale Konstruktion $LL(k)$-Automat

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

3.2 Formale Konstruktion SLL(k)-Automat

1. **Initial** $Q = \{q_0\}$ und $R = \varnothing$, mit $q_0 = [Z \rightarrow \bullet\ S;\#]$.
   Anfangszustand und erster Kellerzustand q_0. Hinweis: $Folge_k(Z) = \{\#\}$.
2. Sei $q = [X \rightarrow \mu \bullet \nu;\ \Omega] \in Q$ und noch nicht betrachtet.
3. **Wenn** $\nu = \varepsilon$ setze $R := R \cup \{q\ \varepsilon \rightarrow \varepsilon\}$
   Auskellern $q'q \rightarrow q'$ mit beliebigem q'.
4. **Wenn** $\nu = \tau\gamma$ mit $\tau \in T$ und $\gamma \in V^*$, setze $q' = [X \rightarrow \mu\ \tau \bullet \gamma;\ \Omega]$.
   Setze $Q := Q \cup \{q'\}$ und $R := R \cup \{q\tau \rightarrow q'\}$.

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

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 $H = \{[B \rightarrow \bullet\, \beta_i; Folge_k(B)] \mid B \rightarrow \beta_i \in P\}$.

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

Setze $Q := Q \cup \{q'\} \cup H$ und $R := R \cup \{q\tau_i \rightarrow q'h_i\tau_i \mid h_i \in H, \tau_i \in Anf_k(\beta_i Folge_k(B))\}$.

3.2 $LL(1)$-Parser mit rekursivem Abstieg

1. Definiere Prozedur X für alle Nichtterminale X
2. Für alternative Produktionen $X \rightarrow X_1 \mid ... \mid X_n$ sei Rumpf von X

```
switch s {
 case Anf(X1 Folge(X)) : Code für X1;
 ...
 case Anf(Xn Folge(X)) : Code für Xn;
 default : Fehler(...);
 }
```

3. Für rechte Seite $X_i = Y_1 \ldots Y_m$ erzeuge:

   $C_1;...;C_m;$ `return;`

   Es gilt $C_i =$

   (a) `if (s ==` Y_i`) s = nächstesSymbol() else Fehler(...);`
   wenn $Y_i \in T$

   (b) Y_i`();` wenn $Y_i \in N$

3.2 Parser aus Grammatik in EBNF

Nichtterminal	*X*	`X();`

Terminal	*t*	`if (symbol == t)`
		`  nextSymbol();`
		`else Fehler();`

Option	[*X*]	`if (symbol ∈ Anf(X)) X();`
Iteration	X^+	`do X();` `while (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(); nextSymbol(); }`
		`else Fehler();`

	%*Z*	`Z();`

Beispielgrammatik

Beispielgrammatik in EBNF-Notation zum Parser auf der nächsten Folie:

Z → A
A → T {'+' T}*
T → F {'*' F}*
F → bez | '(' A ')'

3.2 Parser für Beispielgrammatik

```
AST parse() { s = nextSymbol(); return Z(); }
AST Z() { return A(); }
AST A() {
  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 */
AST F() {
  if (s == bez) { AST r=new AST(s); s=nextSymbol(); return r;}
  if (s == '(') {
    s = nextSymbol();  AST res = A();
    if (s == ')') 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! Parser wird Interpretierer der Tabelle.
  Vorteile: Vermeidung von Prozeduraufrufen, einfachere Fehlerbehandlung. Nachteil: nicht von Hand programmierbar.

Kapitel 3: Syntaktische Analyse

- 1. Theoretische Grundlage: Kontextfreie Grammatiken
  Notation
  Konkrete und abstrakte Syntax
  Kellerautomaten
  Elimination von Linksrekursion und ε-Produktionen
  Systematische Parserkonstruktion: 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-Parser erfassen alle deterministischen kf Grammatiken. Sie sind noch einigermaßen schnell berechenbar und es gibt ausgereifte 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 Fehlermeldung

```
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 die vorige Fehlermeldung verstehen?
  - Zweck aller nachfolgenden Ausführungen ist nur darauf zu beziehen.
  - Detailliertes Wissen über automatische Konstruktion vernachlässigbar
  - Aber: Intuitive „Hand“konstruktion muß beherrscht werden.
- Lösungen:
  - Sprache ändern: abschließendes „end“ einführen (zulässig?)
  - 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 ;
```

3.3 Wdh. Nichtdeterministische LL- und LR-Parser

•Für LL: (prädiktiv)
Vergleich (compare):

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

Vorhersage (produce):

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

top-down Parser

vom Startsymbol zum Wort

•Für LR: (reduzierend)
Schift (shift):

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

Reduktion (reduce):

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

bottom-up Parser

vom Wort zum Startsymbol

Anmerkung: Der Zustand q ist noch bedeutungslos, er wird später zum deterministisch machen benötigt werden.

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

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

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

gilt

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

- 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 Idee für deterministisch Machen des LR-Parsers

- Nimm alle verfügbare Information (Reduktionsklassen),
  - d.h. unendliche Vorschau auf die noch zu verarbeitende Eingabe
  - und den gesamten Kellerinhalt.
- Allerdings sind die Reduktionsklassen nicht effektiv berechenbar
  - Deshalb: Beschränkung auf k-Zeichen Vorschau und
  - Erkenntnis, dass der zur jeweiligen Entscheidung nötige Kellerinhalt in einem bzw. beschränkt vielen Zuständen subsumierbar ist.
  - Dies führt zu k-Kellerklassen
- Mit Hilfe der k-Kellerklassen wird der charakteristische Automat def.
- Aus dem charakteristischen Automaten ist die Übergangsfunktion des LR(k)-Parsers ablesbar.

- Aber: Deterministisch machen geht nur für eingeschränkte Grammatikklassen

3.3 Textmengen (Wdh.)

$k : x = \quad x\#, \qquad$ falls $\qquad |x| < k$

$k : x = \quad x_1 \ldots x_k\,, \quad$ falls $\qquad x = x_1 \ldots x_m \wedge m \geq k$

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

in der Literatur auch $\text{First}_k(x)$ genannt

$\text{Anf}'_k(x) = \qquad \{u \mid u \in \text{Anf}_k(x) \wedge \neg \exists\, A \in N,\ y \in T^* \;$ so daß $x \Rightarrow_R Auy \Rightarrow uy\}$

in der Literatur auch $\text{EFF}_k(x)$ genannt

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

in der Literatur auch $\text{Follow}_k(x)$ genannt

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 $A_p \rightarrow y_p$, $p=1,...,n$:

$$R_0 = \{(r'r, ss') \mid Z \Rightarrow^*_R r'As', A \Rightarrow_{R'} rs, s \neq \varepsilon\}$$

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

$R_i \cap R_j = \varnothing$ 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. $R_i \cap R_j = \varnothing$ nicht algorithmisch entscheidbar.

Problem ist der unbeschränkt lange Eingaberest.

3.3 Warum ist Anf' nötig: Beispiel

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

$P =$ $\quad 1 : Z \rightarrow S,$ $\quad 2: S \rightarrow Ab,$ $\quad 3 : A \rightarrow c,$ $\quad 4: A \rightarrow \varepsilon$

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

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

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

- Beispiel: Rechtsableitung $Z \Rightarrow_R S \Rightarrow_R Ab \Rightarrow_R b$:
  e auf A reduzieren, bevor das Zeichen b geschiftet wird.
- Ohne Unterscheidung zwischen $\Rightarrow_{R'}$ und $\Rightarrow_R$:
  (e, b#) gehört zu R_0, also wird geschiftet aber Konfiguration $bq\#$ kann keiner Reduktionsklasse zugeordnet werden, d.h. Sackgasse

3.3 Situationen

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

Kellerinhalt und Zustand sq mit einem Zustandssymbol codieren

(Prüfung mehrerer Einträge im Keller vermeiden)

Lösungsidee:

- bei LL und LR ist s rechte bzw. linke Seite einer Produktion $X \rightarrow x_1 \ldots x_n$
- Übergänge $tqt \rightarrow q$ (bei LL) bzw. $qt \rightarrow tq$ (bei LR) sind nur zulässig, wenn in der Produktion ein Terminalzeichen t ansteht, $x_1 \ldots x_n = x'tx'''$, wobei $x'' := tx'''$
- also: ersetze sqx durch Situation $[X \rightarrow x' \bullet x'';x]$, die durch den Punkt anzeigt, wieweit 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.
- Situationen heißen englisch *items*.

3.3 k-Kellerklassen

- Idee: Beschränke den betrachteten Eingaberest auf $k \geq 0$ Zeichen: k-Kellerklasse K_p^k, $p=0,\ldots,n$ definiert durch

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

- Vergleich mit LR-Definition: Mit

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

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

- 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 Reguläre Erkenner für k-Kellerklassen

Satz (Büchi):

Alle Kellerklassen K_p^k für $p=0,...,|N|-1$ sind regulär.

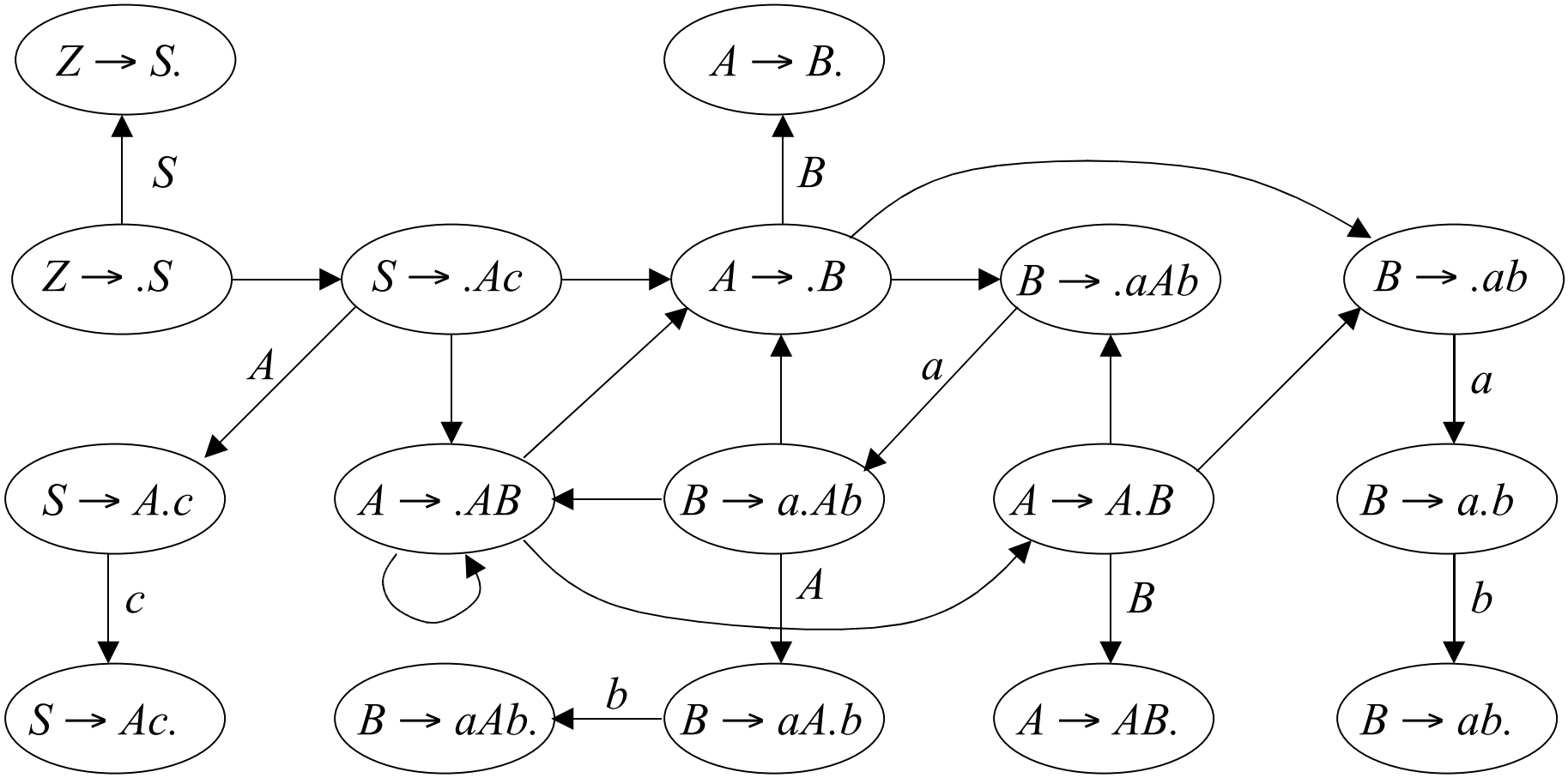
Nachweis durch Angabe einer rechtslinearen Grammatik G_p^k für jede Kellerklasse.

Man kann daraus einen nichtdeterministischen endlichen Automaten C ableiten, der die regulär erkennbaren Wortanfänge der durch die (original) Grammatik G gegebenen Sprache akzeptiert. Dieser hat:

- Situationen als Zustände $\in Q$
- $[Z \rightarrow \bullet\, S; \#]$ als Startzustand (Z nichtrekursives Start-NT von G)
- die Übergangsfunktion f
  - $f([X \rightarrow x \bullet vy\,;\,\omega]\,,\,v) = [X \rightarrow xv \bullet y\,;\,\omega]$ mit $v \in V$
  - $f([X \rightarrow x \bullet By\,;\,\omega]\,,\,\varepsilon) = [B \rightarrow \bullet b\,;\,\tau]$ mit $B \rightarrow b \in P,\ \tau \in \mathrm{Anf}_k(y\omega)$
- Endzustände sind belanglos für das weitere Vorgehen

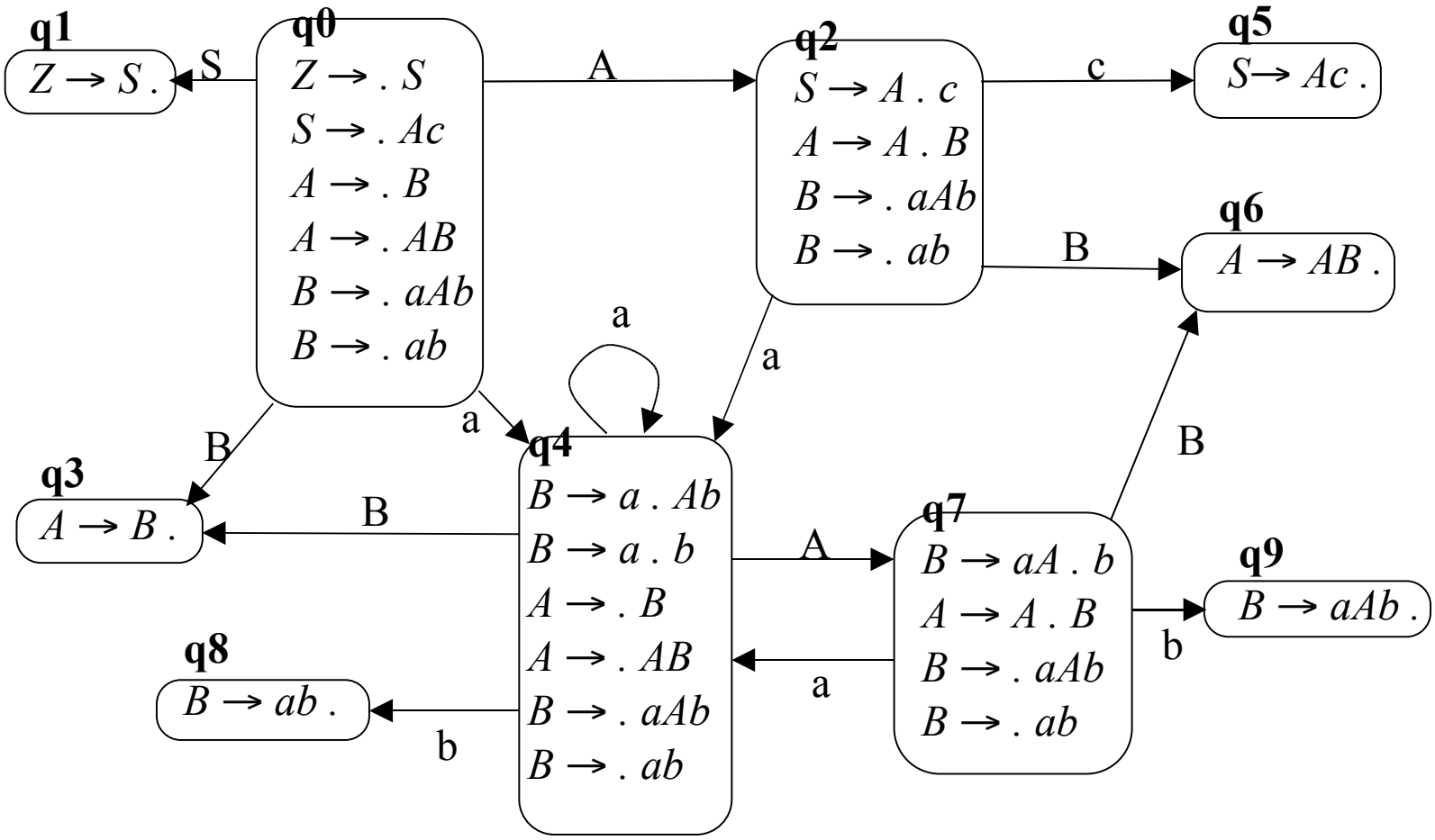
3.3 Nichtdeterministischer, endlicher Automat *C*

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



3.3 Charakteristischer Automat für *k=0*

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



3.3 Hilfskonstruktionen mit dem charakteristischen Automaten

- Der nichtdeterministische endliche Automat C kann mit den bekannten Verfahren deterministisch gemacht werden.
  Ergebnis: charakteristischer Automat

  - Definiere $basis(q,t,\omega)$ als Menge von Situationen die mit Einlesen des (Nicht-)Terminals t aus Situationen von q ausgehend erreichbar sind:

    $$basis(q,t,\omega) = \{[X \rightarrow \mu t \bullet \gamma\ ;\ \omega] \mid [X \rightarrow \mu \bullet t\gamma\ ;\ \omega] \in q\}$$

  - Definiere $next(q,t,\omega)$ als die transitive Hülle von $basis(q,t,\omega)$

    $$next(q,t,\omega) = H(basis\ (q,t,\omega)), \text{ wenn } basis(q,t,\omega) \neq \varnothing$$

    mit $H(M) = M \cup \{[B \rightarrow \bullet \beta\ ;\ \tau] \mid \exists [X \rightarrow \mu \bullet\ B\ \gamma\ ;\ \xi] \in H(M)\ ;$
    $B \rightarrow \beta \in P, \tau \in \mathrm{Anf}_k(\gamma\xi)\}$

    Hinweis: $H(M)$ erweitert Situationen (oder Situtationsmengen) auf die Zustände des charakteristischen Automaten.

3.3 Übergangsfunktion des $LR(k)$-Automaten

Für einen charakteristischen Automaten C, $q \in Q$, $t \in V$ und $v \in T^{\leq k-1}$:

$$f(q, tv) := \begin{cases} next(q,t,\omega) & \text{wenn } [X \to \alpha \bullet t\gamma; \omega] \in q \wedge (v \in \text{Anf}'_{k-1}(\gamma\omega) \vee k < 2) \\ \text{RED}(X \to x) & \text{wenn } [X \to x \bullet ; k{:}tv] \in q \\ \text{HALT} & \text{wenn } t = \# \wedge [Z \to S \bullet ; \#] \in q \\ \text{FEHLER} & \text{sonst} \end{cases}$$

- Zustand q heißt inadäquat, wenn Übergangsfunktion $f(q, tv)$ für irgendein tv nicht eindeutig die Fälle eins, zwei und drei unterscheiden kann. Nur zwei Möglichkeiten existieren:
  - Schift-Reduktionskonflikt: schiften und 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.

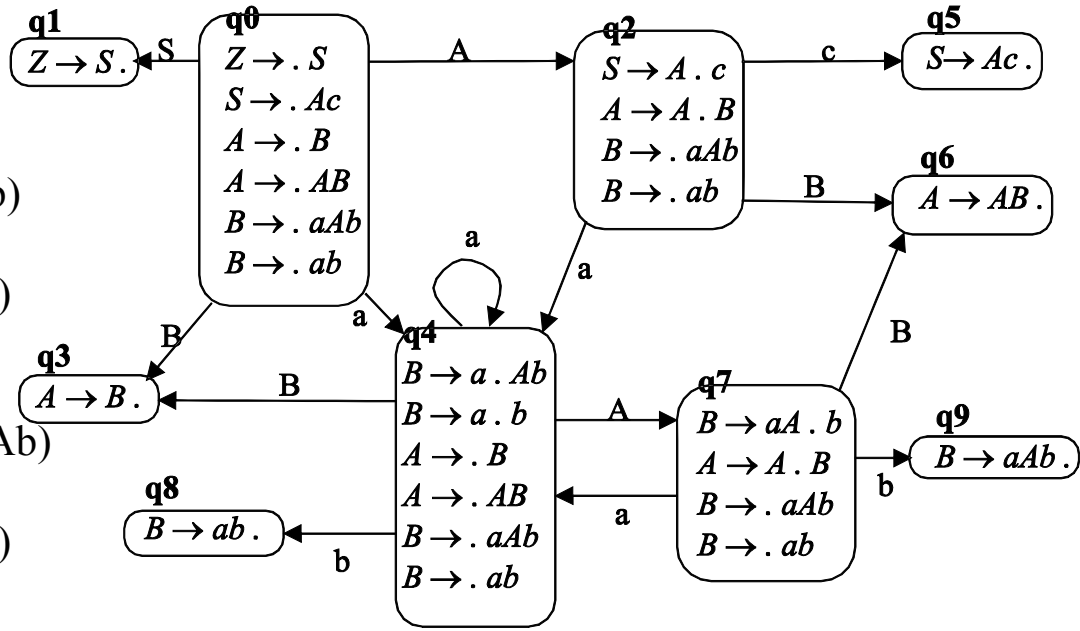
3.3 Ablauf des $LR(k)$-Automaten

- Startzustand des Automaten ist: $H(\{[Z \rightarrow \bullet S\ ;\ \#]\})$
- In jedem Schritt wird die Übergangsfunktion f angewandt:
  - Beim Schiften wird statt des konkreten Symbols ein Zustand des charakteristischen Automaten auf den Keller gelegt. Nur durch Schiften von Terminalen wird die Eingabe verkürzt; Schiften von Nichtterminalen läßt die Eingabe unverändert.
  - $\mathrm{RED}(X \rightarrow x)$ bedeutet Reduzieren mit Produktion $X \rightarrow x$, d.h. :
    - Löschen von |x| Kellereinträgen.
    - Oberster Kellereintrag wird aktueller Zustand.
    - Die Übergangsfunktion $f\,(q,\ Xtv)$ wird angewandt.
      Dies führt sofort zu einem Schift des Nichtterminals X und kann auch in einem Schritt mit dem Kellerlöschen behandelt werden.
  - Bei HALT wird die Eingabe akzeptiert.
  - Bei FEHLER ist die Eingabe nicht akzeptabel.

3.3 Beispiel: Ablauf eines $LR(0)$-Automaten

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

Keller	Eingabe	Bemerkungen
0	aabbc#	schifte(a)
04	abbc#	schifte(a)
044	bbc#	schifte(b)
0448	bc#	reduziere(B → ab)
04	bc#	schifte(B)
043	bc#	reduziere(A → B)
04	bc#	schifte(A)
047	bc#	schifte(b)
0479	c#	reduziere(B → aAb)
0	c#	schifte(B)
03	c#	reduziere(A → B)
0	c#	schifte(A)
02	c#	schifte(c)
025	#	reduziere(S → Ac)
0	#	schifte(S)
01	#	HALT

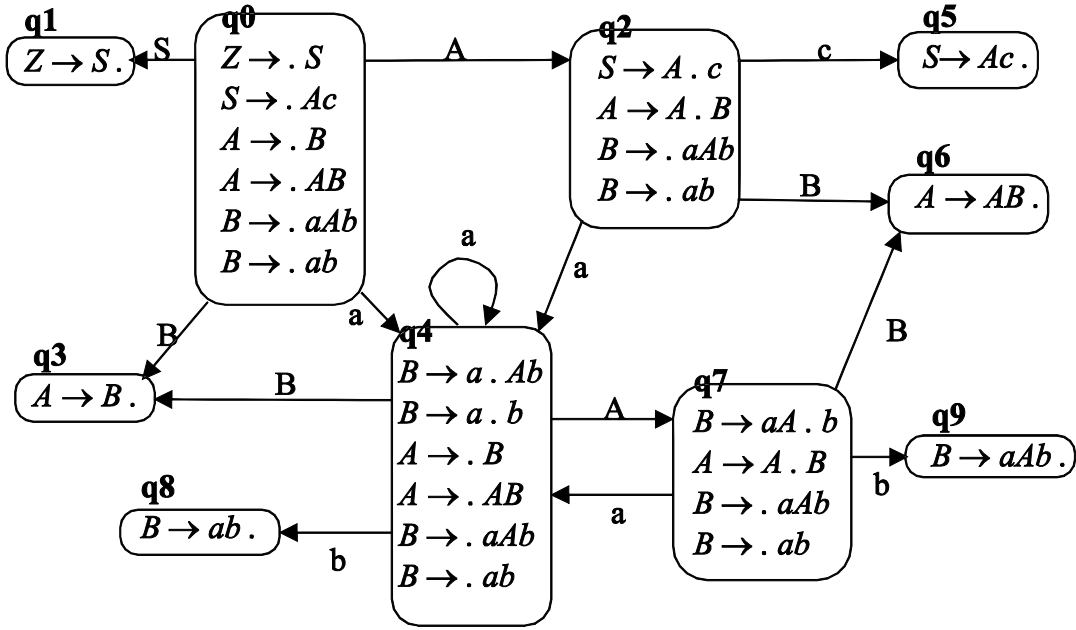


rot: aktuelle Symbole bzw. zu reduzierender Kellerteil
blau: Reduktionssymbol mit Schift im nächsten Schritt

3.3 Beispiel: Übergangstabelle eines $LR(0)$-Automaten

(1) $Z \rightarrow S$, (2) $S \rightarrow Ac$, (3) $A \rightarrow AB$, (4) $A \rightarrow B$, (5) $B \rightarrow aAb$, (6) $B \rightarrow ab$

	a	b	c	#	A	B	S
0	4	-	-	-	2	3	1
1	-	-	-	#	-	-	-
2	4	-	5	-	-	6	-
3	+4	+4	+4	-	+4	+4	+4
4	4	8	-	-	7	3	-
5	+2	+2	+2	-	+2	+2	+2
6	+3	+3	+3	-	+3	+3	+3
7	4	9	-	-	-	6	-
8	+6	+6	+6	-	+6	+6	+6
9	+5	+5	+5	-	+5	+5	+5



–	Fehler
+r	Reduziere mit Regel r
q	Schifte, neuer Zustand: q
#	HALT

3.3 Beispiel: Ein $LR(2)$ Kellerautomat

$$(1)\ Z \rightarrow X,\ (2)\ X \rightarrow Y,\ (3)\ X \rightarrow bYa,\ (4)\ Y \rightarrow c,\ (5)\ Y \rightarrow ca$$

Keller	Eingabe	Bemerkungen
0	bcaa#	schifte(b)
03	caa#	schifte(c)
031	aa#	schifte(a)
0312	a#	reduziere(Y → ca)
03	a#	schifte(Y)
035	a#	schifte(a)
0358	#	reduziere(X → bYa)
0	#	schifte(X)
01	#	HALT

	bc	c#	ca	a#	aa	#	X#	Y#	Ya
0	3	7	7	-	-	-	9	8	-
1	-	-	-	+4	2	-	-	-	-
2	-	-	-	+5	-	-	-	-	-
3	-	-	1	-	-	-	-	-	4
4	-	-	-	5	-	-	-	-	-
5	-	-	-	-	-	+3	-	-	-
6	-	-	-	-	-	+5	-	-	-
7	-	-	-	6	-	+4	-	-	-
8	-	-	-	-	-	+2	-	-	-
9	-	-	-	-	-	#	-	-	-

rot: aktuelle Symbole bzw. zu reduzierender Kellerteil

grün: Vorschau

blau: Reduktionssymbol mit Schift im nächsten Schritt

-	Fehler
+r	Reduziere mit Regel r
q	Schifte, neuer Zustand: q
#	HALT

3.3 Warum nicht LR(1) ?

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.

3.3 $SLR(k)$-Grammatiken

$SLR(k)$-Grammatik (simple LR(k)): Eine Grammatik heißt $SLR(k)$, wenn sie LR(0) ist oder die modifizierte Übergangsfunktion bezüglich des charakteristischen LR(0)-Automaten C, $q \in Q$, $t \in V$ und $v \in T^{\leq k-1}$:

$$f(q, tv) := \begin{cases} next(q,t,\varepsilon) & \text{wenn } [X \to \mu \bullet t\gamma] \in q \wedge (v \in \text{Anf}'_{k-1}(\gamma\text{Folge}_{k-1}(X)) \vee k < 2) \\ \text{RED}(X \to x) & \text{wenn } [X \to x \bullet] \in q,\ k{:}tv \in \text{Folge}_k(X) \\ \text{HALT} & \text{wenn } t = \# \wedge [Z \to S \bullet] \in q \\ \text{FEHLER} & \text{sonst} \end{cases}$$

keine inadäquaten Zustände liefert.

- rot: Unterschied LR(k) und SLR(k)
- Achtung: $next(q,t,\varepsilon)$ operiert auf einem charakteristischen LR(0)-Automaten, also ohne Rechtskontexte.
- Problem: $SLR(k)$ genügt zwar für Ausdrucksgrammatiken, berücksichtigt aber Linkskontext zu wenig (da die Vorausschau ohne Kenntnis der schon verarbeiteten mithin links stehenden Zeichen bestimmt wird).

Kapitel 3: Syntaktische Analyse

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

3.3.1 *LALR*(k)-Grammatiken

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

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 der zerteilbaren Sprachen zwischen LALR(k) und LR(k) ist praktisch unerheblich.
- Alle verbreiteten LR-Parsergeneratoren (yacc, pgs, lalr, bison, ...) konstruieren LALR(1)-Automaten.

3.3.1 LALR(k) Übergangsfunktion

Mit einem charakteristischen Automaten C' in dem alle Zustände mit gleichem Kern zusammengelegt sind, $q \in Q$, $t \in V$ und $v \in T^{\leq k-1}$:

$$f(q, tv) := \begin{cases} next(q,t,v) & \text{wenn } [X \to \mu \bullet t\gamma \,;\, \omega] \in q \wedge (v \in \mathrm{Anf}'_{k-1}(\gamma\omega) \vee k < 2) \\ \mathrm{RED}(X \to x) & \text{wenn } [X \to x \bullet \,;\, k{:}tv] \in q \\ \mathrm{HALT} & \text{wenn } t = \# \text{ und } [Z \to S \bullet \,;\, \#] \in q \\ \mathrm{FEHLER} & \text{sonst} \end{cases}$$

erhalten wir die LALR(k)-Übergangsfunktion.

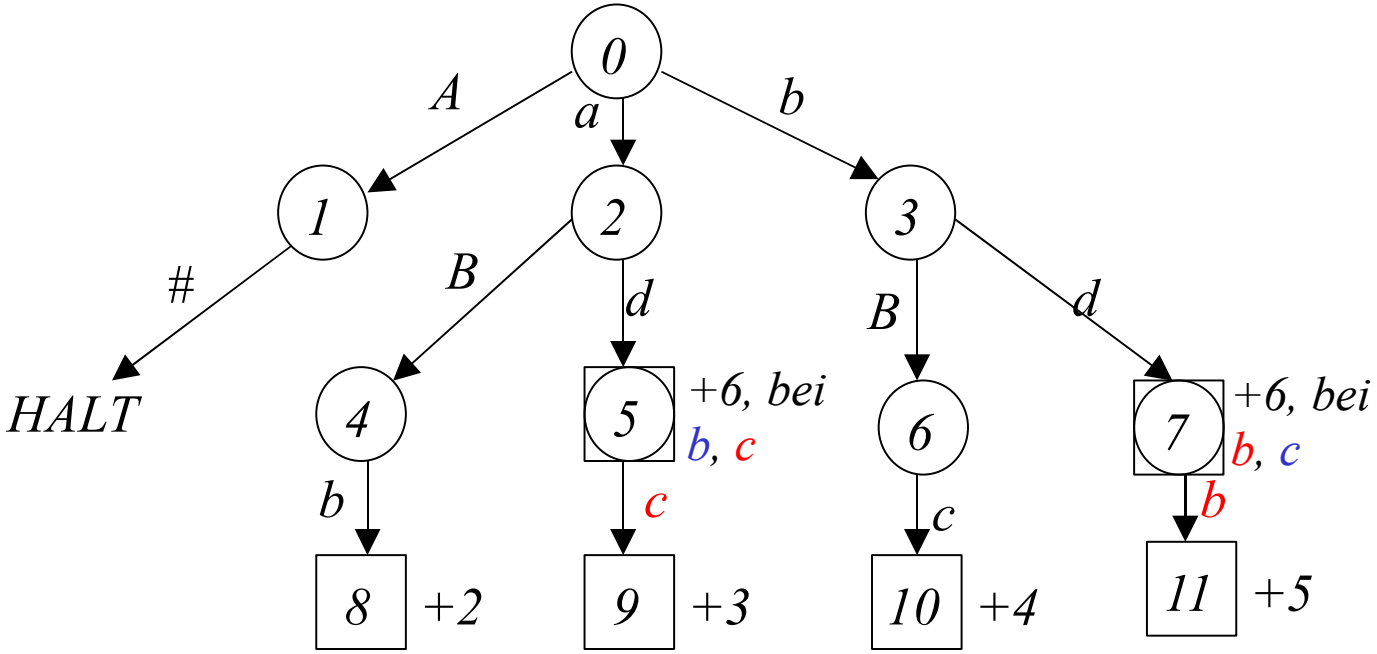
- rot: Unterschied LR(k) und LALR(k)
  Einziger Unterschied: Zusammenlegen der Zustände „modulo Kern“
- Im Unterschied zu SLR(k) benutzt LALR(k) einen „echten“ Rechtskontext von $X \to x$, der schärfer trennt als $\mathrm{Folge}_k(X)$.

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 ;id: und [id:

3.3.1 Behebung inadäquater Zustände

- Parsergenerator:
  - Schift-Reduzier-Konflikt: Schiften wird bevorzugt
  - Reduzier-Reduzier-Konflikt: Zuerst spezifizierte Reduktion wird häufig bevorzugt. Meist fehlerbehaftet.
- Maßnahmen:
  - Automatische Konfliktauflösung auf Korrektheit prüfen!
  - „Faktorisieren“ gemeinsamer Produktionsteile (siehe if-then-else)
  - Verwendung von Präzendenzen (z.B. bei bison)
  - Vergrößern der erkannten Sprache, nachher Einschränken mit semantischer Analyse
  - An Beispielen lernen
  - Mächtigeren Generator (höheres k, LR statt LALR) verwenden, allerdings in der Regel nicht hilfreich

Achtung: Es gibt keine allgemeingültigen Verfahren

3.3.1 LALR(1) Konstruktion am Beispiel

(1) $Z \rightarrow A$

(2) $A \rightarrow A + T$

(3) $A \rightarrow T$

(4) $T \rightarrow T * F$

(5) $T \rightarrow F$

(6) $F \rightarrow bez$

(7) $F \rightarrow (A)$

3.3.1 Anfangszustand

Zustand q_0

Basis	Hülle	Übergang 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

Basis	Hülle	Übergang 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

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

Übergang

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

3.3.1 Zustand 3

Zustand q_3

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

3.3.1 Zustand 4

Zustand q_4

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

3.3.1 Zustand 5

Zustand q_5

Basis	Hülle	Übergang 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 gekennzeichneter Basen erweitert vorhandene Zustände.

3.3.1 Erweitere Zustand 2

Zustand q_2

Basis	Hülle	Übergang 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

$[A \rightarrow T\bullet; \{+\#)\}]$		Reduktion bei +,#,)	
$[T \rightarrow T\bullet * F; \{*+\#)\}]$		*	$\{[T \rightarrow T*\bullet F; \{*+\#)\}]\}$

3.3.1 Erweitere Zustand 3

Zustand q_3

Basis	Hülle	mit	Übergang 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

Basis	Hülle	Übergang 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

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

Basis	Hülle	mit	Übergang 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 gekennzeichneter Basen ergibt keine Erweiterung vorhandener Zustände.

3.3.1 Zustand 7

Zustand q_7

Basis	Hülle	mit	Übergang 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

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

3.3.1 Erweitere Zustand 6

Zustand q_6

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	$\{[F\rightarrow bez\ \bullet;\{*+\#\}]\}$
	$[F\rightarrow \bullet(A);\{*+\#\}]$	$($	$\{[F\rightarrow(\bullet A);\{*+\#\}]\}$

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

Neuer Zustand enthält) in der Folgemenge:

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

Basis	Hülle	mit	Übergang in neue Basis
$[F \rightarrow (A)\bullet;\{*+\#)\}]$			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	$f(q_1,\#) =$ HALT $\; f(q_1,+)=q_6$
q_2	$f(q_2,+\#)\,) =$ Reduziere$(A \rightarrow T) \; f(q_2,*)=q_7$
q_3	$f(q_3,*+\#)\,) =$ Reduziere$(T \rightarrow F)$
q_4	$f(q_4,*+\#)\,) =$ 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	$f(q_9,+\#)\,) =$ Reduziere$(A \rightarrow A+T) \; f(q_9,*)=q_7$
q_{10}	$f(q_{10},*+\#)\,)=$ Reduziere$(T \rightarrow T*F)$
q_{11}	$f(q_{11},*+\#)\,) =$ 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			

–	Fehler,
+i	Reduziere mit Regel r,
i	Schifte, neuer Zustand: i
„“	Don't care

3.3.1 Fakten und Fragen zu LR

- Automat $qt \rightarrow tq,\ t \in T,\ x_1 \ldots x_n q \rightarrow Xq, X \rightarrow x_1 \ldots x_n \in P$ 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(k)-Automat zu groß: SLR(k)-Klasse (Vorschau: $\mathrm{Folge}_k(A)$) ungenügend
- LALR(k): Zustände des LR(k)-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 Beispiele

- Abstrakte Beispiele:
  - **http://www.info.uni-karlsruhe.de/~heuzer/grammars.ps**
- 1. Grammatik:
  - Parser-Generator: **antlr**
  - LL (k) für $k \geq 0$
  - Quellsprache: Java Spec. 2.0 für `kjc/kaffe`
  - Implementierungssprache: Java
- 2. Grammatik:
  - Parser-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: Syntaktische Analyse

- 1. Theoretische Grundlage: Kontextfreie Grammatiken
  Notation
  Konkrete und abstrakte Syntax
  Kellerautomaten
  Elimination von Linksrekursion und ε-Produktionen
  Systematische Parserkonstruktion: 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 unerheblich)
- LR(0)-Reduktionszustände können beseitigt werden, indem im vorigen Zustand bereits reduziert wird (Schift-Reduktionszustand)
- Kettenproduktionen eliminieren
- echte und unechte (don‘t care) Fehlerübergange unterscheiden. Unecht: Übergang kann nie erreicht werden, z.B. alle leeren Übergänge mit Nichtterminalen
- Fehlerübergänge ausfaktorisieren in Fehlermatrix F:
  $f(q,t) =$ **`if`** $F[q,t]$ **`then`** *Fehler* **`else`** *Eintrag_in_Übergangsmatrix*
- Übergangsmatrix komprimieren: leere Übergänge berücksichtigen
- Übergangsmatrix weiter komprimieren (siehe nächste Folie)

3.3.2 Tabellenkompression

Methoden zur weiteren Kompression der Übergangsmatrix:

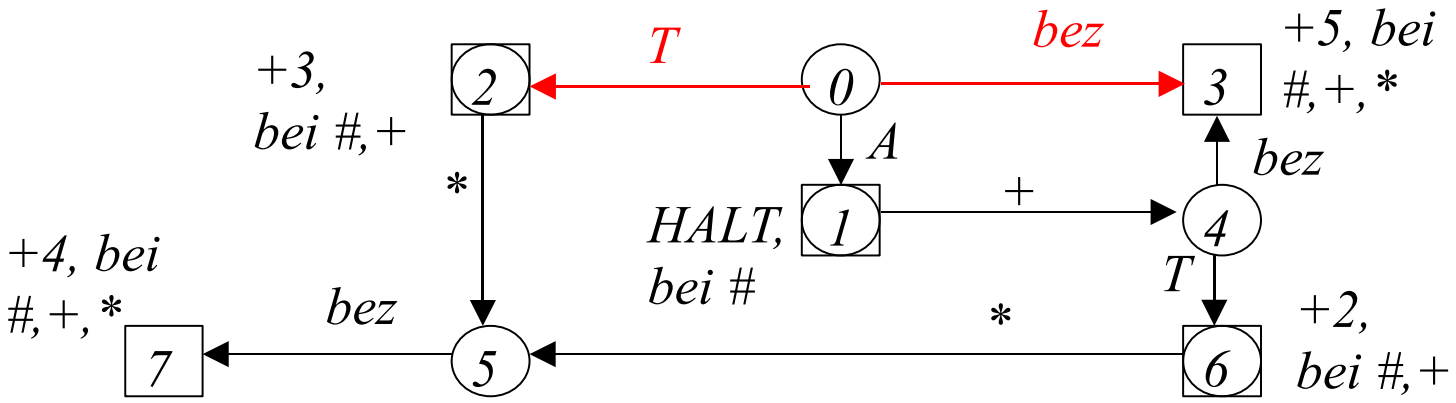
- Graphenfärben: Unverträglichkeitskanten zwischen ungleichen/unverträglichen Tabelleneinträgen. Gleichgefärbte Zeilen/Spalten(-Teile) können zusammengelegt werden.
- Index-Zugriffs-Methoden: Umsortieren und Kombinieren der Einträge, so daß mit konstant vielen Indizierungen der Eintrag findbar ist.
- Listen-Suche: Suche nach dem richtigen Eintrag via Schlüssel und variabler Vergleichsanzahl.
- Hinweise:
  - Verwendbar bei beliebigen dünn/gleichförmig besetzten Tabellen
  - Historisch: Bedingung für Anwendbarkeit (|Tabelle| < Hauptspeicher)
    Heute: Geschwindigkeit (|Tabelle| < Cachegröße)

Quelle: Peter Dencker, Karl Dürre und Johannes Heuft, „Optimization of parser tables for portable compilers“, ACM Trans. Program. Lang. Syst., 6 (4), 1984, Seiten 546-572, http://doi.acm.org/10.1145/1780.1802

3.3.2 Kettenproduktionen

(1) $Z \rightarrow A$ (2) $A \rightarrow A + T$ (3) $A \rightarrow T$

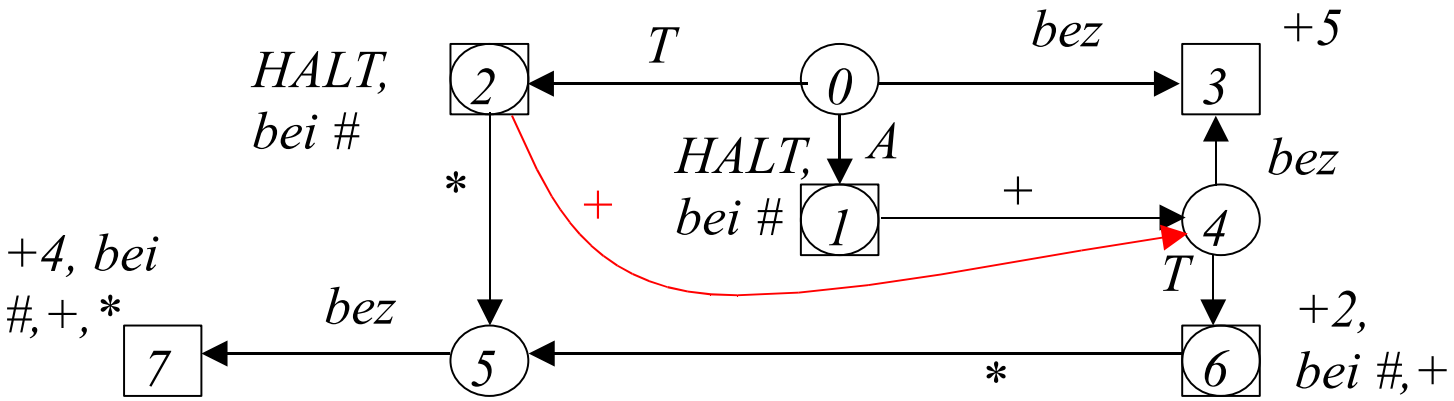
(4) $T \rightarrow T*bez$ (5) $T \rightarrow bez$



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			

-	Error,
+i	Reduziere mit Regel r,
-i	Schifte, reduziere dann mit Regel r
i	Schifte, neuer Zustand: i
„“	Don't care

3.3.2 Fehlermatrix und komprimierte Übergangsmatrix

	A	*T,F*
0,1,2	1	2
3	6	7
4		8
5,6,7,8		-4

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

3.3.2 Tabellengröße

Tabellengröße nach Elimination von Ketten (Beispiel ADA 83):

- 95 Terminale, 252 Nichtterminale
- 540 Zustände (kodiert 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 Parsergeneratoren

	Bison	yacc	PGS	Lalr	Ell
Grammatik	LALR(1)	LALR(1)	LALR(1)	LALR(1)	LL(1)
Spezifiziert in	BNF	BNF	EBNF	EBNF	EBNF
Geschwindigkeit in [10^3 Symbol / Sekunde]	8.93	15.94	17.32	34.94	54.64
Geschwindigkeit in [10^3 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.

3.2.2 Komplexität Parsen in $O(n)$

Für alle Klassen gilt:

- Jeder Ableitungsschritt benötigt, da niemals ein Rücksetzen notwendig ist, konstanten Aufwand $O(1)$.
- Zerteilen benötigt Aufwand $O(n)$
  wobei n die Anzahl der Ableitungsschritte ist.

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

3.3.2 Komplexität

n : Anzahl der Produktionen, k : Länge der Vorausschau

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)	$O(2^{n^{k+1}+k\log n})$	$O(n^{2(k+1)})$

3.3.2 Sätze über kontextfreie Grammatiken

Satz 1: Für jede $\mathrm{LR}(k)$-Grammatik G mit $k > 1$ gibt es eine $LR(1)$-Grammatik G´ mit $L(G) = L(G')$.
Beweis durch Rechtsfaktorisierung.

Satz 2: Jede $\mathrm{LL}(k)$-Grammatik ist auch $\mathrm{LR}(k)$.

Satz 3: Es gibt $\mathrm{LR}(k)$-Grammatiken, die für kein k' $\mathrm{LL}(k')$ 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 $\mathrm{LL}(k)$ oder $\mathrm{LR}(k)$ ist.

Kapitel 3: Syntaktische Analyse

- 1. Theoretische Grundlage: Kontextfreie Grammatiken
  Notation
  Konkrete und abstrakte Syntax
  Kellerautomaten
  Elimination von Linksrekursion und ε-Produktionen
  Systematische Parserkonstruktion: 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 unpassender 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, ...)

Hinweis: Dieser Abschnitt behandelt Fehlermeldungen aller Übersetzerphasen, nicht nur syntaktische!

3.4 Fehlerbehandlung II

Fehler: Nicht (vom Programmierer) intendiertes Programm

Fehlersymptom:

- Sichtbare Auswirkung des Fehlers: Verletzung der Sprachdefinition
- parserdefinierte 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
  - möglichst 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 Parser 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 Parserdefinierte Fehlerstelle

parserdefinierte Fehlerstelle t:

$\forall \underline{s} \in \sum^* : st\underline{s} \notin L$ und $\exists\, s' : ss' \in L$.

LL-, SLL-, LR-, SLR- und *LALR-Parser* finden die parserdefinierte Fehlerstelle.

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

3.4 Beweisskizze für $LL(k)$

Angenommen t parserdefinierte Fehlerstelle, d.h. $st\underline{s} \in L$,

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

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

Terminal t ist parserdefinierte Fehlerstelle gdw:

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

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 parserdefinierter Fehlerstelle $st\underline{s} \notin L$: frühest möglichen Wiederaufsetzpunkt finden
- Totalkorrektur:
  - Eingabe $st\underline{s} = s_1x_1...s_nx_n$, $s_1 = s$, 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 $s_1y_1...s_ny_n$ korrekt ist
  - Nachteil: quadratischer Aufwand, praktisch nicht eingesetzt
- Die Zeichen, an denen wieder aufgesetzt werden kann, bilden die Ankermenge.

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-SYMPTOM 1: } erwartet*/
 else {
  if ( i != j ){
    j: = i ;          /*FEHLER 2 bleibt unerkannt*/
  }                   /*Ende if */
 }                    /*Ende main*/
 return 0 ;           /*FEHLER-SYMPTOM 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 Systematische Korrektur (Röhrich)

gegeben parserdefinierter Fehlerstelle $str \notin L$
gesucht Fortsetzung sy in L:

- Bestimme eine Ankermenge $D=\{ d \in \Sigma \mid y=ss'ds'' \in L \}$
- 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
- Vorteile:
  - nahe bei Minimalkorrektur, 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

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

3.4 ALGOL 60 – Systematische Korrektur (Röhrich)

```
BEGIN
  INTEGER ARRAY A,B(1...5 1...10);
                          ^
***Error in front of 1     <,> inserted

  INTEGER I,J,K,L;
  UP:I+J>K+L*4 THEN GO L1 ELSE K IS 2;
      ^        ^       ^
***Error in front of <+>   <:=> inserted
***Error in front of THEN  <;> inserted
                           THEN deleted
***Error in front of <L1>  TO inserted
                           ELSE <K> IS <2> deleted
```

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 Syntaktische Analyse aufbauen und in getrenntem Keller (Datenstruktur) ablegen oder als Argument bei Prozeduraufruf mitgeben.

3.4 *LL* Beispiel Automat

$Z \rightarrow A$

$A \rightarrow FA`$

$A` \rightarrow \mathsf{e} \mid + FA`$

$F \rightarrow i \mid (A)$

$q_0 : [Z \rightarrow \cdot A\ ]$

$q_1 : [Z \rightarrow A \cdot\ ]$

$q_2 : [Z \rightarrow \cdot FA']$

$q_3 : [A \rightarrow F \cdot A']$

$q_4 : [F \rightarrow \cdot i]$

$q_5 : [F \rightarrow \cdot (A)]$

$q_6 : [A \rightarrow FA' \cdot\ ]$

$q_7 : [A' \rightarrow \cdot\ \mathsf{e}\ ]$

$q_8 : [A` \rightarrow \cdot + F\ A`]$

$q_9 : [F \rightarrow i \cdot\ ]$

$q_{10}\ [F \rightarrow (\cdot\ A)]$

$q_{11}\ [A' \rightarrow + \cdot\ FA']$

$q_{12}\ [F \rightarrow (A \cdot\ )]$

$q_{13}\ [A' \rightarrow + F \cdot\ A']$

$q_{14}\ [F \rightarrow (A)\ \cdot\ ]$

$q_{15}\ [A' \rightarrow + FA' \cdot\ ]$

* zeichnet die vorherzusagenden Produktionen bzw. Zustandsübergänge aus

$^*q_0 i \rightarrow q_1\ q_2\ i,\ q_0(\rightarrow q_1\ q_2(,$

$^*q_1 \rightarrow \mathsf{e},$

$^*q_2 i \rightarrow q_3 q_4 i,\ q_2(\rightarrow q_3 q_5(,$

$^*q_3 \# \rightarrow q_6\ q_7\ \#,\ q_3) \rightarrow q_6 q_7),\ q_{3\rightarrow}\ q_6 q_8 +,$

$^*q_4 i \rightarrow q_9,$

$^*q_5\ (\rightarrow q_{10},$

$^*q_6 \rightarrow \mathsf{e}\ ,$

$^*q_7 \rightarrow \mathsf{e},$

$^*q_8 + \rightarrow q_{11},$

$^*q_9 \rightarrow \mathsf{e},$

$^*q_{10}\ i \rightarrow q_{12}\ q_2\ i,\ q_{10}\ (\rightarrow \mathrm{q}_{12}\ \mathrm{q}_2\ (,$

$^*q_{11} i \rightarrow q_{13}\ q_4 i\ ,\ q_{11}\ (\rightarrow q_{13}\ q_5\ (,\ ^*q_{12}) \rightarrow q_{14},$

$^*q_{13}\ \# \rightarrow\ q_{15}\ q_7\ \#,\ q_{13}\ ) \rightarrow q_{15}\ q_7\ ),\ q_{13}\ + \rightarrow q_{15}\ q_{15} +,$

$^*q_{14}\ \rightarrow \varepsilon\ ,\ ^*q_{15} \rightarrow \varepsilon\ .$

3.4 *LL* Beispiel `i + #`

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

3.4 Problem am Beispiel

Grammatik:

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

SLL-Analyse des fehlerhaften Satzes:

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

mit parserdefiniertem Fehler := landet in Situation

$[I \rightarrow$ **if** $E\bullet$ **then** Z **end;** $\Omega] := b$ **then ... end**

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

3.4 Problem am Beispiel

leicht modifizierte Grammatik:

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

Parsen von: **`if`** a `:=` b **`then ... end`**

mit parserdefiniertem Fehler := landet in Situation

$[I \rightarrow$ **`if`** $E \bullet$ **`then`** Z **`end;`** $\Omega] :=$ b **`then ... end`**

Beachte: `:=` $\notin$ Folge(B)

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

(1) $Z \rightarrow A\ \#$

(2) $A \rightarrow A + F$

(3) $A \rightarrow F$

(4) $F \rightarrow \text{bez}$

(5) $F \rightarrow (\ A\ )$

0: $Z \rightarrow \cdot\ A\ ;\ \#$

$A \rightarrow \cdot\ F\ ;\ \#+$

$A \rightarrow \cdot\ A + F\ ;\ \# +$

$F \rightarrow \cdot\ bez\ ;\ \# +$

$F \rightarrow \cdot\ (A)\ ;\ \# +$

1: $Z \rightarrow A\ \cdot ;\ \#$

$A \rightarrow A \cdot + F\ ;\ \# +$

2: $A \rightarrow F\ \cdot ;\ \#+\ )$

3: $F \rightarrow bez\ \cdot ;\ \# +)$

4: $F \rightarrow (\cdot\ A\ )\ ;\ \#+\ )$

$A \rightarrow \cdot\ F\ ;\ )+$

$A \rightarrow \cdot\ A + F\ ;\ ) +$

$F \rightarrow \cdot\ bez\ ;\ )+$

$F \rightarrow \cdot\ (A)\ ;\ ) +$

5: $A \rightarrow A + \cdot\ F\ ;\ \# +)$

$F \rightarrow \cdot\ bez\ ;\ \# +)$

$F \rightarrow \cdot (A\ )\ ;\ \#+\ )$

6: $F \rightarrow (\ A \cdot)\ ;\ \#+\ )$

$A \rightarrow A \cdot + F\ ;\ ) +$

7: $A \rightarrow A + F\ \cdot ;\ \# +)$

8: $F \rightarrow (A) \cdot\ ;\ \#+\ )$

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 zum Fehler	Fortsetzung finden		Wiederaufnahme
$q_0 i +) \; i \; \#$	$q_0 q_1 q_5$	$D = \{ i \; (\; \}$	$q_0 q_1 q_5 i \; \#$ 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