Kapitel 8

Syntaktische Analyse

8.1 Die Aufgabe der syntaktischen Analyse

Der Parser realisiert die syntaktische Analyse von Programmen. Er erhält die Programme in Form einer vom Scanner/Sieber hergestellten Folge von Symbolen. Seine Aufgabe ist es, in diesen Folgen von Symbolen die syntaktische Struktur der Programme zu finden, d.h. Teilfolgen zu immer größeren syntaktischen Einheiten zusammenzufassen.

Syntaktische Einheiten in imperativen Sprachen sind Variablen, Ausdrücke, Anweisungen, Anweisungsfolgen, Deklarationen, Spezifikationen; in funktionalen Sprachen sind es Variablen, Ausdrücke, Muster, Definitionen und Deklarationen und in logischen Sprachen Variablen, Terme, Listen von Termen, Ziele und Klauseln.

In der konzeptionellen Sicht des Übersetzers erkennt der Parser die syntaktische Struktur eines Programms und stellt diese Struktur geeignet dar, so daß weitere Übersetzerteile damit arbeiten können. Eine mögliche Darstellung ist der Syntaxbaum des Programms. Dieser kann mit statischer semantischer Information dekoriert, zur Effizienzsteigerung transformiert und schließlich in ein Maschinenprogramm übersetzt werden.

Manchmal ist allerdings die Übersetzungsaufgabe für eine Programmiersprache so leicht, daß die Programme in einem Durchgang übersetzt werden können. Dann muß der Parser keine explizite Darstellung der syntaktischen Struktur liefern. Stattdessen fungiert er als Hauptprogramm, welches an geeigneten Stellen der Syntaxanalyse Unterprogramme zur semantischen Analyse und zur Codeerzeugung aufruft.

Zum Erkennen der syntaktischen Struktur kommt noch eine andere wichtige Aufgabe hinzu, nämlich die Erkennung und gute „Behandlung“ von Syntaxfehlern. Die meisten Programme, die einem Übersetzer präsentiert werden, enthalten Fehler, viele davon syntaktische Fehler; diese entstehen durch Schreibfehler, wie z.B. nicht korrekt ausgezählte Klammern. Von jedem Übersetzer wird erwartet, daß er syntaktische Fehler möglichst genau lokalisiert. Dabei kann er i.a. nicht die Fehlerstelle selbst feststellen, sondern nur die früheste Stelle, wo der Fehler zu einer Situation geführt hat, in der keine Fortsetzung der bisher analysierten Eingabe zu einem korrekten Programm möglich ist. Läuft der Übersetzer in Batcharbeitsweise, so erwartet man, daß er nicht nach einem entdeckten Fehler

aufhört, sondern möglichst bald wieder in einen Zustand kommt, in dem er das restliche Programm analysieren bzw. weitere Fehler entdecken kann.

Arbeitet er in einer interaktiven Umgebung, so genügt es, wenn er einen oder wenige Fehler meldet, die Information über die Fehlerstellen an einen angeschlossenen Editor weitergibt, und dieser dann den Programmierer an die Fehlerstellen geleitet.

Die syntaktische Struktur der Programme einer Programmiersprache läßt sich durch eine kontextfreie Grammatik beschreiben. Aus einer solchen Grammatik läßt sich automatisch ein zugehöriger Syntaxanalysator, ein Kellerautomat generieren. Aus Effizienz- und Eindeutigkeitsgründen beschränkt man sich im Übersetzerbau meist auf deterministisch analysierbare kontextfreie Grammatiken. Für sie existieren bewährte Parsergeneratoren. Man kann allerdings, wie wir später sehen werden, einen Parser zu einer gegebenen kontextfreien Grammatik auch (in Form eines Programms) „zu Fuß" schreiben. Dies empfiehlt sich jedoch immer dann nicht, wenn eventuell noch Änderungen an der Syntax der Sprache vorgenommen werden. I.a. ist es leichter, die Eingabegrammatik für den Parsergenerator zu ändern und einen neuen Generierungslauf zu machen, als den geschriebenen Parser zu modifizieren. Dieses Argument ist dann umso zutreffender, wenn auch die Syntaxfehlererkennung und -behandlung automatisch erzeugt wird.

Die in der Praxis eingesetzten Syntaxanalyseverfahren fallen in zwei Klassen. Sie arbeiten deterministisch und lesen Programme von links nach rechts. **Top down-Analysatoren** beginnen die Analyse und die Konstruktion des Syntaxbaums mit dem Startsymbol der Grammatik, der Markierung der Wurzel des Syntaxbaums. Sie machen jeweils eine Voraussage darüber, wie das Programm bzw. Teile des Programms aussehen und versuchen anschließend sie zu bestätigen. Technisch ist diese Voraussage eine Satzform, genauer eine Linkssatzform der Grammatik. Die erste noch vollkommen unbestätigte Prognose besteht also aus dem Startsymbol der Grammatik. Nehmen wir an, ein Teil davon sei durch die bereits gelesene Eingabe schon bestätigt. Beginnt der unbestätigte Teil der Prognose mit einem Nichtterminal, so wählt der top down-Parser mithilfe der nächsten noch nicht konsumierten Eingabesymbole deterministisch eine der Produktionen für dieses Nichtterminal aus und stellt eine neue Prognose auf. Beginnt die aktuelle Prognose mit einem Terminalsymbol, so vergleicht er dies mit dem nächsten Eingabesymbol. Herrscht Übereinstimmung, so ist ein weiteres Symbol der Prognose bestätigt. Andernfalls liegt ein Fehler vor. Fertig ist der top down-Parser, wenn die ganze Eingabe bestätigt ist.

Deterministische **bottom up-Parser** beginnen die Analyse und die Konstruktion des Syntaxbaums mit dem Eingabewort, dem zu analysierenden Programm. Sie versuchen, für immer längere Anfangsstücke der Eingabe die syntaktische Struktur zu finden. Das gelingt ihnen, wenn sie Vorkommen von Produktionen der kontextfreien Grammatik entdecken. Eine gefundene rechte Produktionsseite wird dann zum Nichtterminal der linken Seite „reduziert". Bottom up-Parser führen im Wechsel die Aktionen „Lesen des nächsten Eingabesymbols" und „Ausführen von soviel Reduktionen wie geboten" durch. Ob und wieviele

Reduktionen möglich sind, bestimmt sich aus dem bereits reduzierten Anfangsstück, genauer daraus, wozu es reduziert wurde, und einem Anfangsstück fester Länge der restlichen Eingabe. Fertig ist der bottom up-Parser, wenn er seine ganze Eingabe gelesen und zum Startsymbol reduziert hat.

Die Behandlung von Syntaxfehlern

Die meisten Programme, die ein Übersetzer zu sehen bekommt, sind fehlerhaft. Denn fehlerhafte Programme werden i.a. mehrfach übersetzt, fehlerfreie Programme nur nach Modifikationen und Portierungen auf andere Rechner. Deshalb sollte ein Übersetzer den „Normalfall", das inkorrekte Quellprogramm, auf adäquate Art und Weise behandeln. Lexikalische Fehler und auch Fehler in der statischen Semantik, also etwa Typfehler, lassen sich einfacher lokal diagnostizieren und behandeln. Syntaxfehler, besonders in der Klammerstruktur des Programms, sind schwierig zu diagnostizieren und zu reparieren. In diesem Abschnitt werden wir die erwünschten und die in der Praxis möglichen Reaktionen eines Parsers auf Syntaxfehler beschreiben.

Ein Parser soll nach obiger Aufgabenbeschreibung nicht nur korrekte Programme akzeptieren, sondern angemessen auf syntaktisch inkorrekte Programme reagieren. Die erwünschten Reaktionen des Parsers kann man folgendermaßen klassifizieren:

(1) melde und lokalisiere den Fehler;

(2) diagnostiziere den Fehler;

(3) korrigiere den Fehler;

(4) faß wieder Tritt, um eventuell vorhandene weitere Fehler zu entdecken.

Die erste Reaktion sollte von jedem Parser erwartet werden; kein Syntaxfehler sollte unbemerkt „durchschlüpfen". Allerdings muß man zwei Einschränkungen machen. In der Nähe eines anderen Syntaxfehlers kann ein Fehler unbemerkt bleiben. Die zweite Einschränkung ist gewichtiger. I.a. entdeckt der Parser einen Fehler dadurch, daß für seine aktuelle Konfiguration keine legale Fortsetzung existiert. Dies ist aber möglicherweise nur ein Symptom für einen vorhandenen Fehler und nicht der Fehler selbst.

Beispiel 8.1.1

$a := a * (b + c * d \quad ;$

Fehlersymptom: ')' fehlt

Hier gibt es mehrere Fehlermöglichkeiten; entweder ist die öffnende Klammer zuviel, oder es fehlt eine schließende Klammer hinter c oder hinter d. Die Bedeutung der drei möglichen Ausdrücke ist jeweils anders. □

Bei anderen Klammerfehlern mit überflüssigen oder fehlenden **begin, end, if**, usw. können Fehlerstelle und Stelle des Fehlersymptoms weit voneinander entfernt sein. Allerdings haben die im folgenden betrachteten Parser, LL(k)- wie LR(k)-Parser, die **Eigenschaft des fortsetzungsfähigen Präfixes**:

Verarbeitet der Parser für eine kontextfreie Grammatik G ein Präfix u eines Wortes, ohne einen Fehler zu melden, so gibt es ein Wort w, so daß uw ein Satz von G ist.

Parser mit dieser Eigenschaft melden also Fehler(symptome) zum frühestmöglichen Zeitpunkt. Obwohl wir i.a. nur das Fehlersymptom und nicht den Fehler selbst entdecken können, werden wir in Zukunft meist von Fehlern sprechen. In diesem Sinne erfüllen die im folgenden vorgestellten Parser die Anforderung (1). Sie melden und lokalisieren Syntaxfehler.

Daß die Forderung (2) nicht ganz zu erfüllen ist, dürfte jetzt klar sein; der Parser wird lediglich eine Diagnose des Fehlersymptoms versuchen. Diese sollte zumindest folgende Information enthalten:

- Stelle des Fehlersymptoms im Programm;
- Beschreibung der Parserkonfiguration (Zustand, erwartetes Symbol, gefundenes Symbol) etc.

Um die Forderung (3), Korrektur eines gefundenen Fehlers, zu erfüllen, müßte der Parser die Intention des Programmierers ahnen. Dies ist i.a. nicht möglich. Die nächst realistischere Forderung wäre die nach einer global optimalen Fehlerkorrektur. Diese ist folgendermaßen definiert. Der Parser wird um die Fähigkeit erweitert, jeweils ein Symbol in einem Eingabewort einzusetzen bzw. zu löschen. Die **global optimale** Fehlerkorrektur für ein ungültiges Eingabewort w ist ein Wort w', welches durch eine minimale Zahl solcher Einsetz- und Löschoperationen aus w hervorgeht. Man sagt, w und w' haben den **kleinsten Abstand** voneinander. Solche Verfahren wurden vorgeschlagen, haben aber wegen ihrer Kosten keinen Eingang in die Praxis gefunden.

An ihrer Stelle begnügt man sich meist mit lokalen Lösungen, Einsetzungen oder Ersetzungen, die mindestens die folgende Forderung erfüllen. Solch eine lokale Korrektur soll den Parser aus der Fehlerkonfiguration in eine neue Konfiguration überführen, in der er zumindest das nächste Eingabesymbol lesen kann. Damit ist gesichert, daß der Parser durch diese lokalen Veränderungen nicht in eine Endlosschleife gerät. Dazu soll das Entstehen von Folgefehlern möglichst vermieden werden.

Weshalb zieht man Korrekturen am Kellerinhalt des Parsers nicht in Betracht? Die Eigenschaft des fortsetzungsfähigen Präfixes besagt, daß der Kellerinhalt bis zum Zeitpunkt der Fehlerentdeckung keinen Anlaß zur Klage bot. Deshalb ist es i.a. schwer, eine mögliche Fehlerursache ausgehend von dem Fehlersymptom im Keller zu lokalisieren. Außerdem steuern häufig Parser die semantische Analyse durch das Anstoßen von semantischen Routinen. Veränderungen im Keller des Parsers würden das Rückgängigmachen der Effekte solcher semantischer Routinen verlangen.

Der Aufbau dieses Kapitels

Im Abschnitt 8.2 werden die Grundlagen der Syntaxanalyse aus der Theorie der formalen Sprachen und Automaten dargestellt. Kontextfreie Grammatiken mit ihrem Ableitungsbegriff und Kellerautomaten, die zugehörigen Erkennungsmechanismen, werden behandelt. Ein spezieller, nichtdeterministischer Kellerautomat zu einer gegebenen kontextfreien Grammatik G wird konstruiert, der die von G definierte Sprache akzeptiert. Aus diesem Kellerautomaten werden später deterministische top down- und bottom up-Kellerautomaten abgeleitet. Die Technik der Grammatikflußanalyse wird eingeführt. Mit ihr kann man Eigenschaften von kontextfreien Grammatiken und Attributgrammatiken berechnen.

In den Abschnitten 8.3 und 8.4 werden die top down- und die bottom up-Syntaxanalyse vorgestellt. Dazu werden die entsprechenden Grammatikklassen charakterisiert, Generierungsverfahren beschrieben und Fehlerbehandlungsalgorithmen dargestellt.

8.2 Theoretische Grundlagen

Ebenso wie die lexikalische Analyse basiert die syntaktische Analyse auf der Theorie der Automaten und formalen Sprachen. Der wesentliche Satz ist die Äquivalenz der beiden Mechanismen kontextfreie Grammatik und Kellerautomat in dem Sinne, daß man

(a) zu jeder kontextfreien Grammatik einen Kellerautomaten konstruieren kann, der die von der Grammatik definierte Sprache akzeptiert, und daß

(b) die von einem Kellerautomaten akzeptierte Sprache kontextfrei ist, also eine (sogar effektiv konstruierbare) kontextfreie Grammatik besitzt.

Vielleicht bedarf es noch einer Begründung, weshalb reguläre Ausdrücke nicht für die Beschreibung der Syntax von Programmiersprachen ausreichen. Es ist aus der Theorie der formalen Sprachen bekannt, daß reguläre Ausdrücke nicht dazu geeignet sind, eingebettete Rekursion zu beschreiben. Diese kommt jedoch in Form von geschachtelten Blöcken, Anweisungen und Ausdrücken in Programmiersprachen vor. Deshalb muß man von den regulären Ausdrücken zu den kontextfreien Grammatiken übergehen, die in der Lage sind, solche rekursiven Strukturen zu beschreiben.

In Abschnitt 8.2.1 und 8.2.2 wiederholen wir kurz die notwendigsten Begriffe über kontextfreie Grammatiken und Kellerautomaten. Der hiermit vertraute Leser kann diese Abschnitte überschlagen und mit Abschnitt 8.2.3 fortfahren, wo zu einer kontextfreien Grammatik ein (etwas unüblicher) Kellerautomat definiert wird, der die von ihr definierte Sprache akzeptiert.

8.2.1 Kontextfreie Grammatiken

Mit kontextfreien Grammatiken läßt sich die syntaktische Struktur von Programmen einer Programmiersprache beschreiben. Diese gibt an, wie Programme aus

Teilprogrammen zusammengesetzt sind, genauer, welche elementaren Konstrukte es gibt, und wie komplexe Konstrukte aus anderen Konstrukten zusammengesetzt werden können.

Die folgenden Produktionsregeln etwa beschreiben den Aufbau von Anweisungen in einer Pascal-ähnlichen Sprache, wobei nicht weiter gesagt ist, wie eine Bedingung (*Bed*), ein Ausdruck (*Ausdr*) oder ein Name aussieht.

Beispiel 8.2.1

Anw	→	If_Anw |
		While_Anw |
		Repeat_Anw |
		Proz_Aufruf |
		Wertzuweisung
If_Anw	→	**if** Bed **then** Anw_Folge **else** Anw_Folge **fi** |
		if Bed **then** Anw_Folge **fi**
While_Anw	→	**while** Bed **do** Anw_Folge **od**
Repeat_Anw	→	**repeat** Anw_Folge **until** Bed
Proz_Aufruf	→	Name (Ausdr_Folge)
Wertzuweisung	→	Name := Ausdr
Anw_Folge	→	Anw |
		Anw_Folge; Anw
Ausdr_Folge	→	Ausdr |
		Ausdr_Folge, Ausdr

□

Die erste Regel gibt an, daß es fünf verschiedene Arten von Anweisungen gibt. Die zweite ist folgendermaßen zu lesen: Eine *If_Anweisung* (ein Wort für *If_Anw*) besteht entweder aus dem Wort **if** gefolgt von einem Wort für *Bed*, gefolgt von dem Wort **then**, gefolgt von einer Anweisungsfolge (genauer einem Wort für *Anw_Folge*), gefolgt von dem Wort **else**, gefolgt von einer weiteren Anweisungsfolge, gefolgt von dem Wort **fi**, oder es besteht aus dem Wort **if**, gefolgt von einer Bedingung, dem Wort **then** und einer Anweisungsfolge, gefolgt von dem Wort fi.

Definition 8.2.1 (Kontextfreie Grammatik)
Eine **kontextfreie Grammatik (kfG)** ist ein Quadrupel $G = (V_N, V_T, P, S)$, wobei V_N, V_T disjunkte Alphabete sind, V_N die Menge der **Nichtterminale**, V_T die Menge der **Terminale**, $P \subseteq V_N \times (V_N \cup V_T)^*$ die Menge der **Produktionsregeln** ist, und $S \in V_N$ das **Startsymbol** ist. □

Die Nichtterminale der Grammatik stehen für Mengen von Wörtern, nämlich für die Wörter, die sie produzieren (siehe Definition 8.2.2). Die Terminale sind die Symbole, welche in zu analysierenden Programmen tatsächlich auftreten. Während wir bei der Behandlung der lexikalischen Analyse von einem Alphabet von Zeichen gesprochen haben, das sind in der Praxis die in einem Programm erlaubten **Zeichen** des ASCII- oder EBCDIC- Zeichensatzes, werden wir in diesem Kapitel von **Symbolen** reden. Bei der von uns im einleitenden Kapitel dargestellten Arbeitsteilung zwischen lexikalischer Analyse (Scanner) und syntaktischer Analyse (Parser) erkennt der Scanner gewisse Zeichenfolgen als lexikalische

Einheiten und übergibt sie an den Parser als Symbole. Solche Symbole, z.B. fett gedruckte Schlüsselwörter oder **id** für die Symbolklasse der Identifier werden in Beispielgrammatiken häufig als Elemente des Terminalalphabets vorkommen.

Notation:
In den folgenden Definitionen, Sätzen, Bemerkungen und Algorithmen wird konsequent die folgende Notation benutzt. Lateinische Großbuchstaben, z.B. A, B, C, X, Y, Z stehen für Elemente aus V_N; Kleinbuchstaben am Anfang des lateinischen Alphabets, also $a, b, c, \ldots$, stehen für Terminale, also Elemente aus V_T; Kleinbuchstaben am Ende des lateinischen Alphabets, etwa u, v, w, x, y, z, stehen für Terminalworte, also Elemente aus V_T^*; griechische Kleinbuchstaben, z.B. $\alpha, \beta, \gamma, \varphi, \psi$ stehen für Wörter aus $(V_T \cup V_N)^*$.

Diese notationellen Festlegungen sind im Sinne einer Deklaration in einem Programm zu verstehen. A ist also deklariert als eine Variable vom Typ „Nichtterminal", α als vom Typ „Wort über $V_N \cup V_T$". Dies wird meist nicht mehr explizit hinzugefügt. Wie in Programmiersprachen mit geschachtelten Gültigkeitsbereichen werden ab und zu Zeichen lokal neu definiert. Damit wird die globale Konvention überdeckt.

Die Relation P wurde schon als Menge von Produktionsregeln tituliert; jedes Element (A, α) der Relation schreiben wir als $A \to \alpha$. Mehrere Produktionen $A \to \alpha_1, A \to \alpha_2, \ldots, A \to \alpha_n$ für ein Nichtterminal A schreiben wir als $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$. Die $\alpha_1, \alpha_2, \ldots, \alpha_n$ heißen dann die **Alternativen** für A.

Beispiel 8.2.2
$G_0 =$
$(\{E, T, F\}, \{+, *, (,), \mathrm{id}\}, \{E \to E + T \mid T, T \to T * F \mid F, F \to (E) \mid \mathrm{id}\}, E)$
$G_1 = (\{E\}, \{+, *, (,), \mathrm{id}\}, \{E \to E + E \mid E * E \mid (E) \mid \mathrm{id}\}, E)$ □

Eine Produktionsregel einer kfG beschreibt, wie man durch Ersetzen von linken Seiten (Nichtterminalen) durch rechte Seiten (Wörter aus $(V_T \cup V_N)^*$) aus Wörtern über $V_T \cup V_N$ neue solche Wörter *„produziert"* oder *„ableitet"*. Die Relation *„produziert direkt"* auf $(V_T \cup V_N)^*$ wird induziert von der Relation P.

Definition 8.2.2 (produziert direkt, produziert, Ableitung)
Sei $G = (V_N, V_T, P, S)$ eine kontextfreie Grammatik. φ **produziert** ψ **gemäß G direkt**, i.Z. $\varphi \underset{G}{\Longrightarrow} \psi$, wenn Wörter σ, τ, α und ein Nichtterminal A existieren, so daß $\varphi = \sigma A \tau, \psi = \sigma \alpha \tau$ und $A \to \alpha \in P$. Wir sagen φ **produziert** ψ **gemäß** G (oder auch ψ ist aus φ **gemäß** G **ableitbar**), i.Z. $\varphi \overset{*}{\underset{G}{\Longrightarrow}} \psi$, wenn eine endliche Folge von Wörtern $\varphi_0, \varphi_1, \ldots \varphi_n, (n \geq 0)$ existiert mit der Eigenschaft, daß $\varphi = \varphi_0, \psi = \varphi_n$ und $\varphi_i \underset{G}{\Longrightarrow} \varphi_{i+1}$ für $0 \leq i < n$. $\varphi_0, \varphi_1, \ldots, \varphi_n$ heißt dann eine **Ableitung** von ψ aus φ gemäß G. Wir schreiben im obigen Fall auch $\varphi \overset{n}{\underset{G}{\Longrightarrow}} \psi$. □

Bemerkung: $\overset{*}{\underset{G}{\Longrightarrow}}$ ist die reflexive und transitive Hülle von $\underset{G}{\Longrightarrow}$.

Beispiel 8.2.3 (Fortführung von Beispiel 8.2.2)
$E \underset{G_0}{\Longrightarrow} E+T \underset{G_0}{\Longrightarrow} T+T \underset{G_0}{\Longrightarrow} T*F+T \underset{G_0}{\Longrightarrow} T*\mathbf{id}+T \underset{G_0}{\Longrightarrow} F*\mathbf{id}+T \underset{G_0}{\Longrightarrow}$
$F*\mathbf{id}+F \underset{G_0}{\Longrightarrow} \mathbf{id}*\mathbf{id}+F \underset{G_0}{\Longrightarrow} \mathbf{id}*\mathbf{id}+\mathbf{id}$, also $E \overset{*}{\underset{G_0}{\Longrightarrow}} \mathbf{id}*\mathbf{id}+\mathbf{id}$,
$E \underset{G_1}{\Longrightarrow} E+E \underset{G_1}{\Longrightarrow} E*E+E \underset{G_1}{\Longrightarrow} \mathbf{id}*E+E \underset{G_1}{\Longrightarrow} \mathbf{id}*E+\mathbf{id} \underset{G_1}{\Longrightarrow} \mathbf{id}*\mathbf{id}+\mathbf{id}$, also
$E \overset{*}{\underset{G_1}{\Longrightarrow}} \mathbf{id}*\mathbf{id}+\mathbf{id}$. □

Definition 8.2.3 (definierte Sprache, Satz, Satzform)
Sei $G=(V_N,V_T,P,S)$ eine kfG. Die von G **definierte (erzeugte) Sprache** ist $L(G)=\{u\in V_T^* | S \overset{*}{\underset{G}{\Longrightarrow}} u\}$. Ein Wort $x\in L(G)$ heißt ein **Satz** von G. Ein Wort $\alpha\in(V_T\cup V_N)^*$ mit $S \overset{*}{\underset{G}{\Longrightarrow}} \alpha$ heißt eine **Satzform** von G. □

Beispiel 8.2.4 (Fortführung von Beispiel 8.2.3)
$\mathbf{id}*\mathbf{id}+\mathbf{id} \in L(G_0)$ und $\mathbf{id}*\mathbf{id}+\mathbf{id} \in L(G_1)$; denn E ist das Startsymbol von G_0 bzw. G_1, und Beispiel 8.2.3 zeigte, daß $E \overset{*}{\underset{G_0}{\Longrightarrow}} \mathbf{id}*\mathbf{id}+\mathbf{id}$ und $E \overset{*}{\underset{G_1}{\Longrightarrow}} \mathbf{id}*\mathbf{id}+\mathbf{id}$ gelten. □

Notation:
Wir werden den Index G in $\underset{G}{\Longrightarrow}$ weglassen, wenn G aus dem Kontext klar ist.

Eine kontextfreie Grammatik enthält möglicherweise Nichtterminale, die nicht zur Erzeugung der Sprache beitragen, bei einigen Definitionen und Sätzen dagegen stören. Deshalb wollen wir sie loswerden.

Definition 8.2.4 (unproduktive, unerreichbare NT, reduzierte kfG)
Ein Nichtterminal A heißt **unerreichbar**, wenn es keine Wörter α,β gibt mit $S \overset{*}{\Longrightarrow} \alpha A\beta$. A heißt **unproduktiv**, wenn es kein Wort u gibt mit $A \overset{*}{\Longrightarrow} u$.
Eine kfG G heißt **reduziert**, wenn sie weder unerreichbare noch unproduktive Nichtterminale enthält. □

Ein unerreichbares Nichtterminal kann also in keiner beim Startsymbol beginnenden Ableitung in einer Satzform auftreten, ein unproduktives Nichtterminal kein Terminalwort produzieren. Eliminiert man diese beiden Arten von Nichtterminalen und alle Produktionen, in denen sie vorkommen, so verändert sich offensichtlich die von der Grammatik definierte Sprache nicht. Wir nehmen im folgenden immer an, daß Grammatiken reduziert sind.

Die syntaktische Struktur eines Programms, wie sie sich als Resultat der syntaktischen Analyse ergibt, ist ein Baum. Dieser Baum ist von theoretischem und praktischem Interesse. Er wird benutzt, um Begriffe wie Mehrdeutigkeit zu definieren (siehe Definition 8.2.6) und um Analysestrategien zu beschreiben (siehe Abschnitte 8.3 und 8.4), dient innerhalb von Übersetzern aber auch als Schnittstelle zwischen verschiedenen Übersetzermoduln. Die meisten Verfahren zur Auswertung semantischer Attribute im Kapitel Semantische Analyse arbeiten auf dieser Baumstruktur.

Definition 8.2.5 (Syntaxbaum)
Sei $G=(V_N,V_T,P,S)$ eine kfG. Sei B ein geordneter Baum, d.h. ein Baum, in dem die Ausgangskanten jedes Knoten geordnet sind. Seine Blätter seien markiert mit Symbolen aus $V_T\cup\{\varepsilon\}$ und innere Knoten mit Symbolen aus V_N. B heißt ein **Syntaxbaum** (synonym: **Strukturbaum**) für ein Wort $x\in V_T^*$ und $X\in V_N$ gemäß G, wenn gilt:
(a) Ist n ein beliebiger innerer Knoten, markiert mit dem Nichtterminal A, so sind entweder seine Kinder von links nach rechts markiert mit $N_1,N_2,\ldots,N_k\in V_N\cup V_T$ und $A\to N_1N_2\ldots N_k$ ist eine Produktion in P, oder sein einziges Kind ist markiert mit ε und $A\to\varepsilon$ ist eine Produktion in P.
(b) Das Blattwort von B, d.h. das Wort, das sich durch Konkatenation der Markierungen der Blätter von links nach rechts ergibt, ist x.
(c) Die Wurzel ist markiert mit X. Ein Syntaxbaum für ein Wort x und das Startsymbol S heißt einfach ein Syntaxbaum für x. □

Beispiel 8.2.5 (Fortführung von Beispiel 8.2.2)
In Abbildung 8.1 sind zwei Syntaxbäume gemäß Grammatik G_1 für das Wort $\mathbf{id}*\mathbf{id}+\mathbf{id}$ dargestellt. □

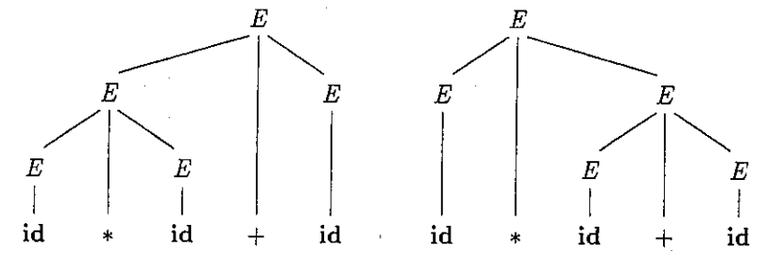

Abb. 8.1:

Definition 8.2.6 (mehrdeutig, eindeutig)
Ein Satz $x\in L(G)$ heißt **mehrdeutig**, wenn er mehr als einen Strukturbaum hat. Eine kfG G heißt **mehrdeutig**, wenn $L(G)$ mindestens einen mehrdeutigen Satz enthält. Eine nicht mehrdeutige kfG nennen wir **eindeutig**. □

Beispiel 8.2.6 (Fortführung von Beispiel 8.2.2)
(1) G_1 ist mehrdeutig, da der Satz $\mathbf{id}*\mathbf{id}+\mathbf{id}$ mehrdeutig ist.
(2) G_0 ist nicht mehrdeutig. □

Beweisskizze für (2):
Man beweise die folgenden Hilfsbehauptungen:

(a) Ist $u \in L(G_0) \cap \{*, \mathbf{id}\}^*$, d.h. u enthält weder '+' noch Klammern, dann gibt es genau einen Syntaxbaum für u und T. Das zeigt man durch Induktion über die Zahl der Vorkommen von '$*$' in u. Der Induktionsschritt geht folgendermaßen: Jeder Syntaxbaum von u und T hat die in Abbildung 8.2 (a) angegebene Gestalt, wenn u mindestens ein '$*$' enthält. t enthält ein Vorkommen von '$*$' weniger. Nach Induktionsannnahme ist t eindeutig bestimmt.
(b) Ist $u \in L(G_0) \cap \{+, *, \mathbf{id}\}^*$, d.h. u enthält keine Klammern, so gibt es genau einen Syntaxbaum für u und E. Wieder benutzt man Induktion über die Anzahl der Vorkommen, diesmal von '+'.
Der Induktionsschritt verläuft ähnlich wie vorher. Jeder Syntaxbaum von u und E hat die in Abbildung 8.2 (b) dargestellte Gestalt, wenn u mindestens ein '+' enthält. Hierbei enthält t_2 kein '+'.
Laut (a) gibt es genau einen Syntaxbaum t_2 für u_2 und T. Wegen der Induktionsannahme gibt es genau einen Baum t_1 für u_1 und E. Zusammen ergibt sich die Behauptung (b).

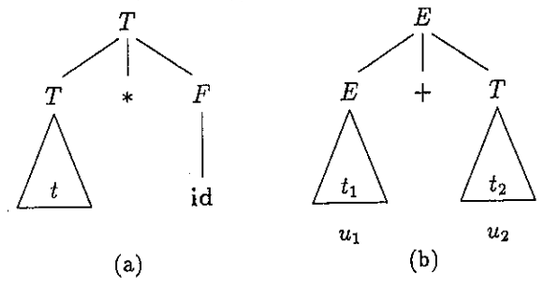

Abb. 8.2:

Jetzt läßt sich zeigen, daß für alle $u \in L(G_0)$ und E genau ein Syntaxbaum existiert. Der Beweis erfolgt durch Induktion über die Anzahl von Klammerpaaren. Im Induktionsschritt nimmt man ein innerstes Klammernpaar her und wendet (b) an. Ersetzt man diesen innersten Klammerausdruck durch **id**, so ergibt sich ein Wort, auf das die Induktionsannahme anwendbar ist. □

Bemerkungen:

1. Jeder Satz x einer Sprache hat mindestens eine Ableitung, genauer gesagt eine Ableitung aus S. Dies ergibt sich aus Definition 8.2.3.

2. Zu jeder Ableitung für einen Satz x gehört ein Syntaxbaum für x.

3. Jeder Satz x besitzt mindestens einen Syntaxbaum. Dies ist eine Konsequenz aus (1) und (2). Es läßt sich auch zeigen durch Angabe eines Verfahrens, das aus einer Ableitung für x einen Syntaxbaum für x konstruiert (siehe Übung 2.1).

4. Zu jedem Syntaxbaum für x gibt es mindestens eine Ableitung für x. Diese Aussage benötigt zum Beweis ein Verfahren, das in der zu (2) entgegengesetzten Richtung arbeitet. Das ist aber einfach anzugeben. □

Beispiel 8.2.7 (Fortführung von Beispiel 8.2.2)
Zu Bem. (1):
Das Wort $\mathbf{id} + \mathbf{id}$ hat gemäß G_1 zwei Ableitungen.
$E \Rightarrow E + E \Rightarrow \mathbf{id} + E \Rightarrow \mathbf{id} + \mathbf{id}$
$E \Rightarrow E + E \Rightarrow E + \mathbf{id} \Rightarrow \mathbf{id} + \mathbf{id}$
Zu Bem. (2):
Zu beiden obigen Ableitungen gehört der in Abbildung 8.3 dargestellte Syntaxbaum.

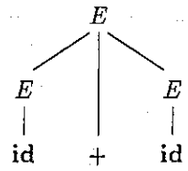

Abb. 8.3:

Hier sieht man, daß der Syntaxbaum die Struktur des Satzes angibt, aber von der eher unwesentlichen Reihenfolge der Anwendungen der Produktionen, in der sich die beiden Ableitungen unterscheiden, abstrahiert. Zu der Ableitung gemäß G_1,
$E \Rightarrow E + E \underset{G_1}{\Rightarrow} E + E + E \underset{G_1}{\Rightarrow} \mathbf{id} + E + E \underset{G_1}{\Rightarrow} \mathbf{id} + \mathbf{id} + E \underset{G_1}{\Rightarrow} \mathbf{id} + \mathbf{id} + \mathbf{id}$,
gehören die beiden verschiedenen Syntaxbäume, die in Abbildung 8.4 dargestellt sind.

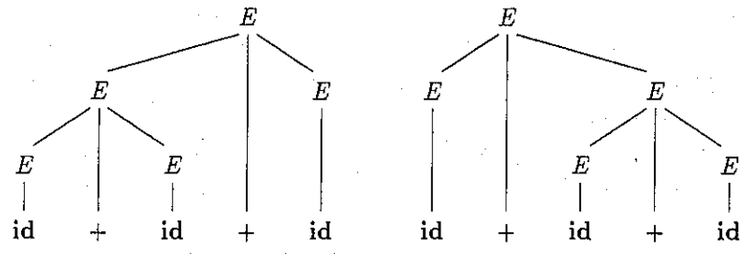

Abb. 8.4:

Man sieht der obigen Ableitung nicht an, welches der Vorkommen von E im 2. Ableitungsschritt ersetzt wird. So können wir, wenn auch nur im Fall mehrdeutiger Grammatiken, in die mißliche Situationen geraten, daß zu einer Ableitung zwei verschiedene syntaktische Strukturen gehören. □

Wir haben in obigem Beispiel gesehen, daß – auch bei eindeutigen Wörtern – zu einem Syntaxbaum mehrere Ableitungen korrespondieren können. Diese ergeben sich aus den verschiedenen Möglichkeiten, in einer Satzform ein Nichtterminal für die nächste Anwendung einer Produktion auszuwählen. Wenn festgelegt wird, daß jeweils das am weitesten links bzw. am weitesten rechts stehende Nichtterminal ersetzt wird, so erhalten wir ausgezeichnete Ableitungen, nämlich die sogenannten Links- bzw. Rechtsableitungen.

Definition 8.2.7 (Linksableitung, Rechtsableitung)
Sei $\varphi_1, \varphi_2, \ldots, \varphi_n$ eine Ableitung von $\varphi = \varphi_n$ aus $\varphi_1 = S$. $\varphi_1, \varphi_2, \ldots, \varphi_n$ heißt eine **Linksableitung** von φ, i.Z. $S \overset{*}{\underset{lm}{\Longrightarrow}} \varphi$, wenn beim Schritt von φ_i nach φ_{i+1} jeweils das in φ_i am weitesten links stehende Nichtterminal ersetzt wird, d.h. $\varphi_i = uA\tau$, $\varphi_{i+1} = u\alpha\tau$ und $A \to \alpha \in P$.

$\varphi_1, \varphi_2, \ldots, \varphi_n$ heißt **Rechtsableitung** von φ, i.Z. $S \overset{*}{\underset{rm}{\Longrightarrow}} \varphi$, wenn jeweils das am weitesten rechts stehende Nichtterminal ersetzt wird, d.h. $\varphi_i = \sigma Au$, $\varphi_{i+1} = \sigma\alpha u$ und $A \to \alpha \in P$.

Eine Satzform, die in einer Linksableitung (Rechtsableitung) auftritt, heißt **Linkssatzform (Rechtssatzform)**. □

Bemerkungen
5. Zu jedem Syntaxbaum gibt es genau eine Links- und genau eine Rechtsableitung.
6. Zu jedem eindeutigen Satz gibt es genau eine Links- und genau eine Rechtsableitung. Dies ist eine Konsequenz aus der Definition der Mehrdeutigkeit und aus Bemerkung 5. □

Beispiel 8.2.8 (Fortführung von Beispiel 8.2.2)
Zu Bemerkung 5.:
Das Wort $\mathbf{id} * \mathbf{id} + \mathbf{id}$ hat gemäß G_1 die Linksableitungen
$E \underset{lm}{\Longrightarrow} E + E \underset{lm}{\Longrightarrow} E * E + E \underset{lm}{\Longrightarrow} \mathbf{id} * E + E \underset{lm}{\Longrightarrow} \mathbf{id} * \mathbf{id} + E \underset{lm}{\Longrightarrow} \mathbf{id} * \mathbf{id} + \mathbf{id}$
und $E \underset{lm}{\Longrightarrow} E * E \underset{lm}{\Longrightarrow} \mathbf{id} * E \underset{lm}{\Longrightarrow} \mathbf{id} * E + E \underset{lm}{\Longrightarrow} \mathbf{id} * \mathbf{id} + E \underset{lm}{\Longrightarrow} \mathbf{id} * \mathbf{id} + \mathbf{id}$.
Es hat die Rechtsableitungen
$E \underset{rm}{\Longrightarrow} E + E \underset{rm}{\Longrightarrow} E + \mathbf{id} \underset{rm}{\Longrightarrow} E * E + \mathbf{id} \underset{rm}{\Longrightarrow} E * \mathbf{id} + \mathbf{id} \underset{rm}{\Longrightarrow} \mathbf{id} * \mathbf{id} + \mathbf{id}$
und $E \underset{rm}{\Longrightarrow} E * E \underset{rm}{\Longrightarrow} E * E + E \underset{rm}{\Longrightarrow} E * E + \mathbf{id} \underset{rm}{\Longrightarrow} E * \mathbf{id} + \mathbf{id} \underset{rm}{\Longrightarrow} \mathbf{id} * \mathbf{id} + \mathbf{id}$.
Zu Bemerkung 6.:
Das Wort $\mathbf{id} + \mathbf{id}$ hat in G_1 nur jeweils eine Linksableitung, nämlich
$E \underset{lm}{\Longrightarrow} E + E \underset{lm}{\Longrightarrow} \mathbf{id} + E \underset{lm}{\Longrightarrow} \mathbf{id} + \mathbf{id}$, und eine Rechtsableitung, nämlich
$E \underset{rm}{\Longrightarrow} E + E \underset{rm}{\Longrightarrow} E + \mathbf{id} \underset{rm}{\Longrightarrow} \mathbf{id} + \mathbf{id}$. □

Wird ein Wort einer eindeutigen Grammatik abgeleitet, und zwar einmal durch eine Links- und einmal durch eine Rechtsableitung, so wissen wir, daß die in den beiden Ableitungen verwendeten Mengen von Produktionen gleich sind. Sie werden lediglich in einer anderen Reihenfolge angewendet. Die Frage ist, kann man in beiden Ableitungen jeweils Satzformen finden, die dadurch miteinander korrespondieren, daß jeweils im nächsten Schritt das gleiche Vorkommen eines Nichtterminals ersetzt wird? Das folgende Lemma stellt eine solche Beziehung her.

Lemma 8.2.1 (a) Wenn gilt $S \overset{*}{\underset{lm}{\Longrightarrow}} uA\varphi$, dann gibt es ein ψ, mit $\psi \overset{*}{\Longrightarrow} u$, so daß für alle v mit $\varphi \overset{*}{\Longrightarrow} v$ gilt: $S \overset{*}{\underset{rm}{\Longrightarrow}} \psi Av$.
(b) Wenn gilt $S \overset{*}{\underset{rm}{\Longrightarrow}} \psi Av$, dann gibt es ein φ mit $\varphi \overset{*}{\Longrightarrow} v$, so daß für alle u mit $\psi \overset{*}{\Longrightarrow} u$ gilt: $S \overset{*}{\underset{lm}{\Longrightarrow}} uA\varphi$. □

Abb. 8.5 verdeutlicht die Zusammenhänge zwischen φ und v und ψ und u .

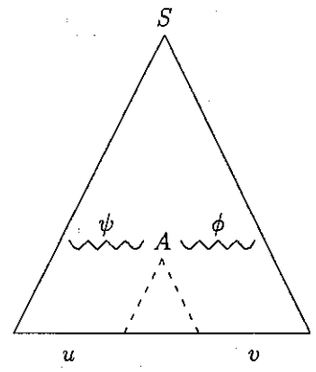

Abb. 8.5: Zusammenhang zwischen Rechts- und Linksableitung

Von kontextfreien Grammatiken, welche die Syntax von Programmiersprachen beschreiben, verlangt man, daß sie eindeutig sind. Dann gibt es zu jedem syntaktisch korrekten Programm, d.h. zu jedem Satz der Grammatik, genau einen Syntaxbaum, genau eine Links- und genau eine Rechtsableitung.

8.2.2 Kellerautomaten

In diesem Abschnitt werden wir den zur Klasse der kontextfreien Grammatiken gehörenden Akzeptor behandeln, den Kellerautomaten. Die Äquivalenz der beiden Konzepte wurde bereits am Anfang von Abschnitt 8.2 erwähnt. Die eine der beiden Richtungen, nämlich die Konstruierbarkeit eines Kellerautomaten zu einer beliebigen kontextfreien Grammatik „riecht“ schon fast nach dem, was wir für

den Übersetzerbau brauchen. Wenn die Syntax einer Programmiersprache kontextfrei beschrieben ist, könnten wir mit einem solchen konstruktiven Verfahren einen Akzeptor für die Programmiersprache gewinnen.

Tatsächlich werden wir in Abschnitt 8.2.3 die Konstruktion eines Kellerautomaten zu einer beliebigen kfG angeben, der die von der kfG definierte Sprache akzeptiert. Allerdings hat dieser Kellerautomat noch Schönheitsfehler; er ist i.a. nichtdeterministisch, auch dann, wenn die Ausgangsgrammatik – wie die in der Praxis verwendeten Grammatiken – einen deterministischen Akzeptor besitzt.

In den Abschnitten 8.3 und 8.4 wird beschrieben, wie – ausgehend von diesem nichtdeterministischen Kellerautomaten – für gewisse Unterklassen der kontextfreien Grammatiken deterministische Automaten erzeugt werden können.

Ein Kellerautomat verfügt im Gegensatz zu den im letzten Kapitel behandelten endlichen Automaten über eine unbegrenzte Speicherfähigkeit. Er besitzt nämlich einen Keller, das ist ein potentiell unendlich langes Feld von Zellen, für welches eine last-in-first-out-Disziplin für das Speichern und Entnehmen gilt. Die bildliche Darstellung eines Kellerautomaten findet sich in Abbildung 8.6.

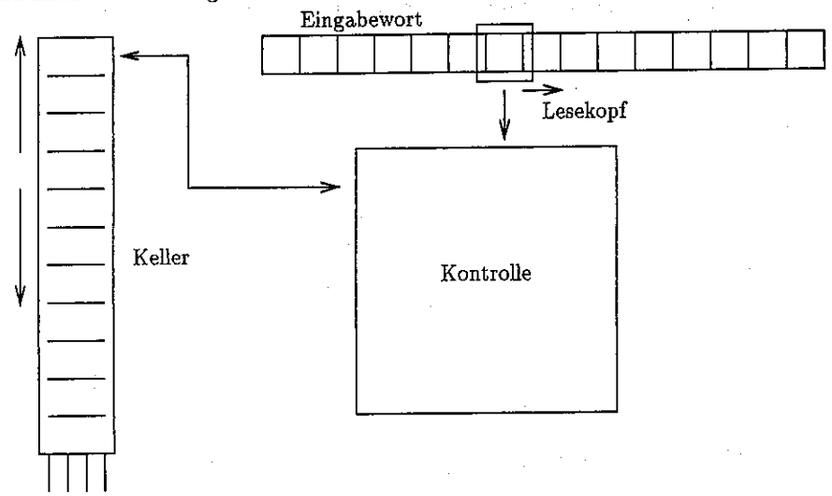

Abb. 8.6: Ein Kellerautomat

Der Lesekopf darf sich nur von links nach rechts bewegen; ein Übergang des Automaten wird bestimmt durch die obersten Kellersymbole und evtl. das aktuelle Eingabesymbol. Der Übergang verändert den Inhalt des Kellers an dessen oberen Ende.

Definition 8.2.8 (Kellerautomat)
Ein **Kellerautomat** (KA) ist ein Tupel $P = (V, Q, \Delta, q_0, F)$, wobei

- V das **Eingabealphabet** und
- Q die endliche Menge der **Zustände**,

- $q_0 \in Q$ der **Anfangszustand** und
- $F \subseteq Q$ die Menge der **Endzustände** sind.
- Δ ist eine endliche Relation zwischen $Q^+ \times (V \cup \{\varepsilon\})$ und Q^*, die **Übergangsrelation**. Δ kann man also betrachten als eine endliche partielle Funktion δ von $Q^+ \times (V \cup \{\varepsilon\})$ in die endlichen Teilmengen von Q^*. □

Beachten Sie, daß diese Definition etwas ungewöhnlich ist, da die Mengen der Zustände und der Kellersymbole identifiziert worden sind. Der Inhalt des Kellers ist also immer eine Folge von Zuständen. Den obersten Zustand im Keller werden wir den **aktuellen** Zustand nennen. Die Übergangsrelation beschreibt die Berechnungen des Kellerautomaten. Sie gibt für endlich viele Kombinationen von nichtleeren Wörtern über Q und Eingabezeichen aus V oder dem leeren Wort ε jeweils eine Menge von Fortsetzungsmöglichkeiten an. Die Fortsetzungsmöglichkeiten sind jeweils Folgen von Zuständen, welche die angeschaute Folge von Zuständen im Keller ersetzen. Wird das aktuelle Eingabezeichen angeschaut, so wird es auch konsumiert, d.h. der Lesekopf wird ein Feld nach rechts gerückt. Übergänge, welche das nächste Eingabezeichen nicht anschauen, heißen **ε-Übergänge**.

In einem Kellerautomaten gibt es mehrere Quellen für nichtdeterministisches Verhalten. Einmal kann es zu einem Paar $(\gamma, a) \in Q^+ \times V$ mehrere Fortsetzungen gemäß Δ geben. Daneben konkurrieren mit dem Paar (γ, a) Paare $(\gamma', a) \in Q^+ \times V$, wobei γ' ein Suffix von γ ist, und eventuell vorhandene Paare (γ', ε); d.h., wenn ein Übergang für (γ, a) auf eine Konfiguration des Automaten paßt, dann passen auch sie.

Wie üblich wird für einen Automatentyp der Begriff der Konfiguration so definiert, daß er alle Komponenten umfaßt, welche für die zukünftigen Schritte des Automaten relevant sind. Das sind bei unserer Art von Kellerautomaten der Kellerinhalt und die restliche Eingabe.

Definition 8.2.9 (Konfiguration)
Eine **Konfiguration** des KA P ist ein Paar $(\gamma, w) \in Q^+ \times V^*$. Ein **Übergang** von P wird durch die binäre Relation $\vdash_P$ zwischen Konfigurationen dargestellt, die folgendermaßen definiert ist:

$(\gamma, aw) \vdash_P (\gamma', w)$ genau dann, wenn $\gamma = \gamma_1\gamma_2$, $\gamma' = \gamma_1\gamma_3$ und $(\gamma_2, a, \gamma_3) \in \Delta$ für $\gamma, \gamma_1, \gamma_3 \in Q^*$, $\gamma_2 \in Q^+$, $a \in (V \cup \{\varepsilon\})$.

Wir definieren wie üblich für Konfigurationen C und C', $C \vdash_P^n C'$, wenn es Konfigurationen $C_1, \ldots, C_{n+1}$ gibt, so daß $C_1 = C$, $C_{n+1} = C'$ und $C_i \vdash_P C_{i+1}$ für $1 \leq i \leq n$ gilt. $\vdash_P^*$, die reflexive transitive Hülle von $\vdash_P$, steht dann für $\bigcup_{n \geq 0} \vdash_P^n$, $\vdash_P^+$ für $\bigcup_{n \geq 1} \vdash_P^n$. (q_0, w) für beliebiges $w \in V^*$, heißt eine **Anfangskonfiguration**, (q, ε), für $q \in F$, eine **Endkonfiguration**. Ein Wort $w \in V^*$ wird **akzeptiert** von P, wenn gilt $(q_0, w) \vdash_P^* (q, \varepsilon)$ für ein $q \in F$. Die **Sprache** von P, $L(P)$, ist die Menge der von P akzeptierten Wörter:
$L(P) = \{w \in V^* \mid w \text{ wird von } P \text{ akzeptiert}\}$. □

Man beachte, daß ein Wort von einem Kellerautomaten akzeptiert wird, wenn es *mindestens* eine Folge von Konfigurationen gibt, die von der entsprechenden Anfangskonfiguration zu einer Endkonfiguration führt. Es kann allerdings auch mehrere erfolgreiche Folgen und auch viele erfolglose geben, die nur einen Anfang des Wortes lesen können. Da man in der Praxis diese Folgen nicht durch Probieren herausbekommen möchte, sind dort nur deterministische Kellerautomaten von Bedeutung.

Definition 8.2.10 (deterministischer Kellerautomat)
Ein Kellerautomat P heißt **deterministisch** (DKA), wenn gilt:
Für $(\gamma_1, a, \gamma_2), (\gamma_1', a', \gamma_2') \in \Delta$ mit γ_1' ist Suffix von γ_1 oder umgekehrt und a' ist Präfix von a oder umgekehrt, folgt $\gamma_1 = \gamma_1', a = a', \gamma_2 = \gamma_2'$. □

Damit sind alle Konkurrenzen zwischen Übergängen ausgeschlossen. Zu jeder Konfiguration gibt es höchstens einen Übergang, und für ein akzeptiertes Wort gibt es genau eine Folge von Konfigurationen.

8.2.3 Der Item-Kellerautomat einer kontextfreien Grammatik

In diesem Abschnitt interessiert uns *eine* Richtung der Äquivalenz zwischen kontextfreien Grammatiken und Kellerautomaten. Es wird ein Verfahren angegeben, mit dem zu jeder kontextfreien Grammatik ein (nichtdeterministischer) Kellerautomat konstruiert werden kann, der die von der Grammatik definierte Sprache akzeptiert.

Dieser Automat ist – weil nichtdeterministisch – nicht für praktische Zwecke interessant, sondern mehr von didaktischem Interesse; ausgehend von ihm können wir durch klare Entwurfsentscheidungen (statt durch Magie) einmal die LL-Analysatoren von Abschnitt 8.3, und zum anderen die LR-Analysatoren von Abschnitt 8.4 ableiten.

Eine entscheidende Rolle spielt der Begriff des kontextfreien Items. Wie immer, wenn wir von Items reden, betrachten wir syntaktische Muster einer bestimmten Art – hier sind es kontextfreie Produktionen – und kennzeichnen, wieweit das Vorliegen eines Musters bereits bestätigt wurde. Die Interpretation, die zu einem kontextfreien Item $[A \to \alpha.\beta]$ gehört, ist die folgende:
„Beim Versuch, ein Wort für A zu erkennen, wurde bereits ein Wort für α erkannt."

Definition 8.2.11 (kontextfreies Item)
Sei G eine kfG, $A \to \alpha\beta$ eine Produktion von G. Dann heißt das Tripel (A, α, β) ein **kontextfreies Item** von G. Wir schreiben das Item (A, α, β) als $[A \to \alpha.\beta]$. α heißt die **Geschichte** des Items. Ein Item $[A \to \alpha.]$ heißt **vollständig**. Mit It_G bezeichnen wir die Menge aller kontextfreien Items von G. Ist $\rho \in It_G^*$, also eine Folge von Items $[A_1 \to \alpha_1.\beta_1]\,[A_2 \to \alpha_2.\beta_2] \ldots [A_n \to \alpha_n.\beta_n]$, so bezeichne $hist(\rho)$ die Konkatenation der Geschichten der Items von ρ, also $hist(\rho) = \alpha_1\alpha_2 \ldots \alpha_n$. □

Nun definieren wir den zu einer kontextfreien Grammatik gehörenden Item-Kellerautomaten. Seine Zustände und damit seine Kellersymbole sind Items der

Grammatik. Der aktuelle Zustand ist das Item, an dessen rechter Seite der Automat gerade arbeitet. Im Keller darunter stehen die Items, von deren rechten Seiten die Bearbeitung bereits begonnen, aber noch nicht vollendet wurde.

Vorher wollen wir aber für „sauberes" Terminieren der konstruierten Kellerautomaten sorgen. Was wären die Kandidaten für die Endzustände? Natürlich alle vollständigen Items $[S \to \alpha.]$, wenn S das Startsymbol der Grammatik ist. Tritt S auch auf der rechten Seite einer Produktion auf, so kommen solche vollständigen Items oben auf dem Keller vor, ohne daß der Automat deswegen terminieren sollte, da darunter noch unvollständig bearbeitete Items liegen. Wir erweitern deshalb die Grammatik um ein neues Startsymbol S', was dann nicht auf einer rechten Seite vorkommt, und eine Produktion $S' \to S$, die die Grundlage der Anfangs- bzw. Endzustände des Item-Kellerautomaten bildet. Die im folgenden auftretenden Grammatiken sind auf diese Art erweitert.

Definition 8.2.12 (Item-Kellerautomat einer kfG)
Sei $G = (V_N, V_T, P, S)$ eine erweiterte kfG. Der Kellerautomat
$K_G = (V_T, It_G, \delta, [S' \to .S], \{[S' \to S.]\})$ heißt der **Item-Kellerautomat** zu G.
Dabei besteht δ aus allen Übergängen der folgenden drei Typen:
(E) $\delta([X \to \beta.Y\gamma], \varepsilon) = \{[X \to \beta.Y\gamma][Y \to .\alpha] \mid Y \to \alpha \in P\}$
(L) $\delta([X \to \beta.a\gamma], a) = \{[X \to \beta a.\gamma]\}$
(R) $\delta([X \to \beta.Y\gamma][Y \to \alpha.], \varepsilon) = \{[X \to \beta Y.\gamma]\}$.
Übergänge gemäß (E) heißen **Expansionsübergänge**, solche gemäß (L) **Leseübergänge** und solche gemäß (R) **Reduktionsübergänge**.

□

Die obige Interpretation kontextfreier Items hängt nicht ab von irgendeinem Kontext, in dem die Items auftreten. Im Item-Kellerautomaten treten sie als Zustände im Keller auf. Ein Item als Zustand in einer Konfiguration des Item-Kellerautomaten bzw. eine Folge von Items als Kellerinhalt können wir noch präziser interpretieren. Die folgende Invariante (I) über Konfigurationen des Item-Kellerautomaten ist gleichzeitig die wesentliche Hilfsbehauptung für den Beweis, daß $L(K_G) \subseteq L(G)$ ist.

(I) Es gelte $([S' \to .S], uv) \vdash_{K_G}^* (\rho, v)$. Dann gilt $hist(\rho) \overset{*}{\underset{G}{\Longrightarrow}} u$.

Die nun folgende Betrachtung versucht, die Arbeitsweise des Automaten K_G zu erklären und gleichzeitig einen Induktionsbeweis (über die Länge von Berechnungen) zu führen, daß die Invariante (I) für jede aus einer Anfangskonfiguration erreichbare Konfiguration erfüllt ist.

Betrachten wir die Anfangskonfiguration für die Eingabe w daraufhin, ob sie die Invariante erfüllt. Diese Anfangskonfiguration ist $([S' \to .S], w)$. Nichts, also ε, wurde bereits gelesen, $hist([S' \to .S]) = \varepsilon$, und es gilt $\varepsilon \overset{*}{\Longrightarrow} \varepsilon$.

Betrachten wir Expansionsübergänge. Die aktuelle, aus der Anfangskonfiguration $([S' \to .S], uv)$ erreichte Konfiguration sei $(\rho[X \to \beta.Y\gamma], v)$. Diese Konfiguration erfüllt nach Induktionsvoraussetzung die Invariante (I); d.h. es gilt $hist(\rho)\beta \overset{*}{\Longrightarrow} u$.

Das Item $[X \to \beta.Y\gamma]$ als aktueller Zustand legt nahe, einen Präfix von v aus Y abzuleiten. Dazu sollte der Automat die Alternativen für Y nichtdeterministisch versuchen. Gerade das beschreiben die Übergänge gemäß (E). Alle möglichen Folgekonfigurationen $(\rho[X \to \beta.Y\gamma][Y \to .\alpha], v)$ für $Y \to \alpha \in P$ erfüllen ebenfalls die Invariante (I); denn es gilt $hist(\rho[X \to \beta.Y\gamma][Y \to .\alpha]) = hist(\rho)\beta \overset{*}{\Longrightarrow} u$.

Die nächsten betrachteten Übergänge sind die Leseübergänge. Die aktuelle, aus $([S' \to .S], uav)$ erreichte Konfiguration, sei $(\rho[X \to \beta.a\gamma], av)$. Nach Induktionsvoraussetzung erfüllt sie (I), d.h. es gilt $hist(\rho)\beta \overset{*}{\Longrightarrow} u$. Es gilt dann $hist(\rho)\beta a \overset{*}{\Longrightarrow} ua$. Damit erfüllt die Nachfolgekonfiguration $(\rho[X \to \beta a.\gamma], v)$ ebenfalls (I).

Bei einem Reduktionsübergang liegt eine von der Anfangskonfiguration $([S' \to .S], uv)$ erreichte Konfiguration $(\rho[X \to \beta.Y\gamma][Y \to \alpha.], v)$ vor. Nach Induktionsvoraussetzung erfüllt sie (I), d.h. es gilt $hist(\rho)\ \beta\alpha \overset{*}{\underset{G}{\Longrightarrow}} u$, wenn u die bereits konsumierte Eingabe ist.

Der aktuelle Zustand ist das vollständige Item $[Y \to \alpha.]$. Seine Bearbeitung wurde (in der Form $[Y \to .\alpha]$) begonnen, als $[X \to \beta.Y\gamma]$ aktueller Zustand war und die Alternativen von Y ausprobiert werden mußten. Die eine Alternative $Y \to \alpha$ kann also jetzt als erfolgreich „abgehakt" werden.

Die Nachfolgekonfiguration $(\rho[X \to \beta Y.\gamma], v)$ erfüllt ebenfalls die Invariante (I); denn aus $hist(\rho)\beta\alpha \overset{*}{\underset{G}{\Longrightarrow}} u$ folgt natürlich auch $hist(\rho)\beta Y \overset{*}{\underset{G}{\Longrightarrow}} u$. □

Satz 8.2.1 *Sei $G = (V_N, V_T, P, S)$ eine kontextfreie Grammatik. Dann gilt $L(K_G) = L(G)$.*

Beweis:

„$\subseteq$"

$L(K_G) = \{w \in V_T^* \mid ([S' \to .S], w) \overset{*}{\vdash}_{K_G} ([S' \to S.], \varepsilon)\}$

(I) besagt, daß $S \overset{*}{\underset{G}{\Longrightarrow}} w$ gilt, wenn $w \in L(K_G)$ ist. Damit ist w auch in $L(G)$.

„$\supseteq$"

Man zeigt durch Induktion über m, die Länge der Ableitung, die folgende Behauptung.

Gilt $A \overset{m}{\underset{G}{\Longrightarrow}} w$ für $m \geq 1$, $A \in V_N$ und $w \in V_T^*$, dann gibt es eine Produktion $A \to \alpha$, so daß $(\rho[A \to .\alpha], wv) \overset{*}{\vdash}_{K_G} (\rho[A \to \alpha.], v)$ für beliebige $\rho \in It_G^*$ und beliebige $v \in V_T^*$ gilt. □

Beispiel 8.2.9

Sei $G' = (\{S, E, T, F\}, \{+, *, (,), \mathbf{id}\}, P, S)$ die Erweiterung von G_0, wobei $P = \{S \to E, E \to E{+}T \mid T, T \to T{*}F \mid F, F \to (E) \mid \mathbf{id}\}$ ist. Die Übergangsrelation Δ von K_{G_0} ist in Tabelle 8.1 dargestellt. □

Beispiel 8.2.10

Eine Konfigurationsfolge von K_{G_0}, die zur Akzeptanz des Wortes $\mathbf{id} + \mathbf{id} * \mathbf{id}$ führt, ist die in Tabelle 8.2 dargestellte. □

Tabelle 8.1: Tabellarische Darstellung der Übergangsrelation aus Beispiel 8.2.9. Die mittlere Spalte gibt die konsumierte Eingabe an.

oberes Kellerende	Eingabe	neues oberes Kellerende
$[S \to .E]$	ε	$[S \to .E][E \to .E+T]$
$[S \to .E]$	ε	$[S \to .E][E \to .T]$
$[E \to .E+T]$	ε	$[E \to .E+T][E \to .E+T]$
$[E \to .E+T]$	ε	$[E \to .E+T][E \to .T]$
$[F \to (.E)]$	ε	$[F \to (.E)][E \to .E+T]$
$[F \to (.E)]$	ε	$[F \to (.E)][E \to .T]$
$[E \to .T]$	ε	$[E \to .T][T \to .T*F]$
$[E \to .T]$	ε	$[E \to .T][T \to .F]$
$[T \to .T*F]$	ε	$[T \to .T*F][T \to .T*F]$
$[T \to .T*F]$	ε	$[T \to .T*F][T \to .F]$
$[E \to E+.T]$	ε	$[E \to E+.T][T \to .T*F]$
$[E \to E+.T]$	ε	$[E \to E+.T][T \to .F]$
$[T \to .F]$	ε	$[T \to .F][F \to .(E)]$
$[T \to .F]$	ε	$[T \to .F][F \to .\mathbf{id}]$
$[T \to T*.F]$	ε	$[T \to T*.F][F \to .(E)]$
$[T \to T*.F]$	ε	$[T \to T*.F][F \to .\mathbf{id}]$
$[F \to .(E)]$	(	$[F \to (.E)]$
$[F \to .\mathbf{id}]$	$\mathbf{id}$	$[F \to \mathbf{id}.]$
$[F \to (E.)]$	)	$[E \to (E).]$
$[E \to E.+T]$	+	$[E \to E+.T]$
$[T \to T.*F]$	*	$[T \to T*.F]$
$[T \to .F][F \to \mathbf{id}.]$	ε	$[T \to F.]$
$[T \to T*.F][F \to \mathbf{id}.]$	ε	$[T \to T*F.]$
$[T \to .F][F \to (E).]$	ε	$[T \to F.]$
$[T \to T*.F][F \to (E).]$	ε	$[T \to T*F.]$
$[T \to .T*F][T \to F.]$	ε	$[T \to T.*F]$
$[E \to .T][T \to F.]$	ε	$[E \to T.]$
$[E \to E+.T][T \to F.]$	ε	$[E \to E+T.]$
$[E \to E+.T][T \to T*F.]$	ε	$[E \to E+T.]$
$[T \to .T*F][T \to T*F.]$	ε	$[T \to T.*F]$
$[E \to .T][T \to T*F.]$	ε	$[E \to T.]$
$[F \to (.E)][E \to T.]$	ε	$[F \to (E.)]$
$[F \to (.E)][E \to E+T.]$	ε	$[F \to (E.)]$
$[E \to .E+T][E \to T.]$	ε	$[E \to E.+T]$
$[E \to .E+T][E \to E+T.]$	ε	$[E \to E.+T]$
$[S \to .E][E \to T.]$	ε	$[S \to E.]$
$[S \to .E][E \to E+T.]$	ε	$[S \to E.]$

Tabelle 8.2: Die Konfigurationenfolge von K_G, die zur Akzeptanz von **id** + **id** $*$ **id** führt. Alle anderen Konfigurationenfolgen führen nicht zur Akzeptanz dieses Wortes.

Kellerinhalt	restliche Eingabe
$[S \to .E]$	**id** + **id** $*$ **id**
$[S \to .E][E \to .E+T]$	**id** + **id** $*$ **id**
$[S \to .E][E \to .E+T][E \to .T]$	**id** + **id** $*$ **id**
$[S \to .E][E \to .E+T][E \to .T][T \to .F]$	**id** + **id** $*$ **id**
$[S \to .E][E \to .E+T][E \to .T][T \to .F][F \to .\mathbf{id}]$	**id** + **id** $*$ **id**
$[S \to .E][E \to .E+T][E \to .T][T \to .F][F \to \mathbf{id}.]$	+**id** $*$ **id**
$[S \to .E][E \to .E+T][E \to .T][T \to F.]$	+**id** $*$ **id**
$[S \to .E][E \to .E+T][E \to T.]$	+**id** $*$ **id**
$[S \to .E][E \to E.+T]$	+**id** $*$ **id**
$[S \to .E][E \to E+.T]$	**id** $*$ **id**
$[S \to .E][E \to E+.T][T \to .T*F]$	**id** $*$ **id**
$[S \to .E][E \to E+.T][T \to .T*F][T \to .F]$	**id** $*$ **id**
$[S \to .E][E \to E+.T][T \to .T*F][T \to .F][F \to .\mathbf{id}]$	**id** $*$ **id**
$[S \to .E][E \to E+.T][T \to .T*F][T \to .F][F \to \mathbf{id}.]$	$*$**id**
$[S \to .E][E \to E+.T][T \to .T*F][T \to F.]$	$*$**id**
$[S \to .E][E \to E+.T][T \to T.*F]$	$*$**id**
$[S \to .E][E \to E+.T][T \to T*.F]$	**id**
$[S \to .E][E \to E+.T][T \to T*.F][F \to .\mathbf{id}]$	**id**
$[S \to .E][E \to E+.T][T \to T*.F][F \to \mathbf{id}.]$	
$[S \to .E][E \to E+.T][T \to T*F.]$	
$[S \to .E][E \to E+T.]$	
$[S \to E.]$	

Kellerautomat mit Ausgabe

Die bisher diskutierten Kellerautomaten sind nur Akzeptoren, d.h. sie entscheiden nur, ob ein vorgelegtes Wort ein Satz der Sprache ist oder nicht. Benutzt man einen Kellerautomaten zur syntaktischen Analyse in einem Übersetzer, so interessiert einen aber auch die syntaktische Struktur akzeptierter Wörter. Diese kann man in Form von Syntaxbäumen, Ableitungen oder auch Folgen von in einer Rechts- bzw. Linksableitung angewandten Produktionen darstellen. Deshalb wollen wir Kellerautomaten jetzt mit Ausgabemöglichkeiten versehen.

Definition 8.2.13 (Kellerautomat mit Ausgabe)
Ein **Kellerautomat mit Ausgabe** ist ein Tupel $P = (V, Q, O, \Delta, q_0, F)$, wobei V, Q, q_0, F wie bisher definiert sind und O ein endliches Ausgabealphabet ist. Δ ist jetzt eine endliche Relation zwischen $Q^+ \times (V \cup \{\varepsilon\})$ und $Q^* \times (O \cup \{\varepsilon\})$.
Eine **Konfiguration** ist ein Element aus $Q^+ \times V^* \times O^*$. □

Bei jedem Übergang kann der Automat ein Zeichen aus O ausgeben. Setzen wir einen Kellerautomaten mit Ausgabe als Parser ein, so besteht sein Ausgabealphabet aus den Produktionen der kfG oder ihren Nummern.

Definition 8.2.14 (Linksparser)
Ein **Linksparser** für eine kfg $G = (V_N, V_T, P, S)$ ist ein Item-Kellerautomat mit Ausgabe $K_G^l = (V_T, It_G, P, \Delta_l, [S' \to .S], \{[S' \to S.]\})$, wobei Δ_l bei den (E)-Übergängen als Ausgabe die angewandte Produktion ausgibt; d.h.
$\Delta_l([X \to \beta.Y\gamma], \varepsilon) = \{([X \to \beta.Y\gamma][Y \to .\alpha], Y \to \alpha) \mid Y \to \alpha \in P\}$. Die anderen Übergänge geben nichts aus.
Eine **Konfiguration** eines Linksparsers ist ein Element aus $It_G^+ \times V_T^* \times P^*$. Die Ausgabe bei einem Expansionsübergang wird rechts an die bereits produzierte Ausgabe angehängt.
$(\rho[X \to \beta.Y\gamma], w, o) \vdash_{K_G^l} (\rho[X \to \beta.Y\gamma][Y \to .\alpha], w, o(Y \to \alpha))$ □

Definition 8.2.15 (Rechtsparser)
Ein **Rechtsparser** für eine kfg $G = (V_N, V_T, P, S)$ ist ein Item-Kellerautomat mit Ausgabe $K_G^r = (V_T, It_G, P, \Delta_r, [S' \to .S], \{[S' \to S.]\})$, wobei Δ_r bei den (R)-Übergängen als Ausgabe die angewandte Produktion angibt, also
$\Delta_r([X \to \beta.Y\gamma][Y \to \alpha.], \varepsilon) = ([X \to \beta Y.\gamma], Y \to \alpha)$. Die anderen Übergänge geben nichts aus. Bei einer Reduktion wird die angewandte Produktion hinten an die bereits produzierte Ausgabe angefügt.
$(\rho[X \to \beta.Y\gamma][Y \to \alpha.], w, o) \vdash_{K_G^r} (\rho[X \to \beta Y.\gamma], w, o(Y \to \alpha))$ □

Die von einem Linksparser ausgegebene Produktionenfolge entspricht der Folge der Ersetzungen in einer Linksableitung. Die Ausgabe eines Rechtsparsers ist die gespiegelte Folge der Regeln, die in einer Rechtsableitung angewendet würden.

Deterministische Parser

Eine entscheidende Eigenschaft des Item-Kellerautomaten haben wir schon mit Satz 8.2.1 bewiesen, daß nämlich der Item-Kellerautomat K_G zu einer kontextfreien Grammatik G genau deren Sprache $L(G)$ akzeptiert.

Eine weitere, eher unangenehme Eigenschaft ist, daß er dies nichtdeterministisch tut. Wo liegt die Quelle für den Nichtdeterminismus von K_G? Wie man leicht sieht, bei den Übergängen des Typs (E). Das Problem ist, daß K_G raten muß, welche der Alternativen er für das aktuelle Nichtterminal, das ist das Nichtterminal hinter dem Punkt, auswählen soll. Bei einer nicht mehrdeutigen Grammatik kann ja höchstens eine der Alternativen angewendet werden, um einen Präfix der restlichen Eingabe abzuleiten.

In den Abschnitten 8.3 und 8.4 werden wir versuchen, diesen Nichtdeterminismus auf zwei Arten zu beseitigen. Die LL-Analysatoren von Abschnitt 8.3 wählen deterministisch eine Alternative für das aktuelle Nichtterminal aus und benutzen dazu eine beschränkte Vorausschau in die restliche Eingabe. Ist also die Grammatik von einem bestimmten Typ, genannt LL(k), so kann genau ein (E)-Übergang ausgewählt werden, wenn man die bereits gelesene Eingabe, das zu expandierende Nichtterminal und die nächsten k Eingabesymbole betrachtet. Dies gilt natürlich nicht für fehlerhafte Eingaben. Da existiert in irgendeiner Konfiguration kein Übergang. LL-Analysatoren sind Linksparser.