2 Attributierte Grammatiken

Ein wichtiges formales Modell für die Übersetzung einer formalen Sprache (und damit natürlich auch der Festlegung einer Semantik für diese Sprache) ist die attributierte Grammatik.
Erste Ideen führen zurück auf eine Arbeit von Irons [32], die Originalarbeit stammt von Knuth [34] aus dem Jahr 1968.

2.1 Definitionen

Die grundlegende Idee der attributierten Grammatik ist es, den Symbolen der Grammatik „Attribute“ (Eigenschaften oder Werte) und den Produktionen Regeln zur Berechnung der Werte der Attribute zuzuordnen.

Definition: Sei $G = (N, T, P, S)$ eine kontextfreie Grammatik. Jedem Symbol $X \in (N \cup T)$ ist eine endliche Menge von **Attributen** $\mathcal{A}(X)$ zugeordnet. Das Attribut $a \in \mathcal{A}(X)$ wird auch mit $X.a$ bezeichnet. Jedem Attribut $a \in \mathcal{A}(X)$ ist eine Wertemenge $\mathcal{W}(X.a)$ zugeordnet. Das Attribut $a \in \mathcal{A}(X)$ kann nur Werte aus seiner Wertemenge $\mathcal{W}(X.a)$ annehmen.
Die Menge der Attribute $\mathcal{A}(X)$ ist disjunkt zerlegt in

- die Menge der **inheriten Attribute** $\mathcal{A}_I(X)$ und in
- die Menge der **synthetischen Attribute** $\mathcal{A}_S(X)$,

d.h. es gilt $\mathcal{A}(X) = \mathcal{A}_I(X) \cup \mathcal{A}_S(X)$ und $\mathcal{A}_I(X) \cap \mathcal{A}_S(X) = \emptyset$.

Die Werte der Attribute eines Symbols X repräsentieren später Eigenschaften dieses Symbols X bezüglich des Ableitungsbaumes, in dem dieses Symbol auftritt.

- Inherite Attribute repräsentieren Aspekte der Umgebung von X im Ableitungsbaum
- Synthetische Attribute repräsentieren Aspekte der weiteren Ableitung von X, d.h. des Ableitungsbaumes unter X.

Bemerkung: Häufig wird in der Theorie außerdem gefordert, dass das Startsymbol S kein inherites Attribut und jedes $X \in T$ kein synthetisches Attribut besitzt. Es hat sich jedoch als vorteilhaft erwiesen, für praktische Probleme auf diese Einschränkung zu verzichten und statt dessen eine Initialisierung von Atributwerten (etwa durch den Scanner) anzunehmen!
In [1] wird eine attributierte Grammatik auch **Syntax-Directed Definition (SSD)** genannt.

Jeder Produktion $X \to Y_1 \dots Y_k$ in P, $X \in N$, $Y_i \in (N \cup T)$, $1 \leq i \leq k$, $k \geq 0$, ist eine endliche Menge von Funktionen (**Regeln**) zugeordnet, die die Berechnung der Werte der synthetischen Attribute von X und der inheriten Attribute aller Y_i festlegen (**Semantische Funktionen**, **semantische Regeln**).
Dabei gilt:

- Für jedes Attribut $a \in \mathcal{A}_S(X)$ gibt es eine Regel, die den Wert von a in Abhängigkeit von Werten anderer Attribute $D(X.a) \subseteq \mathcal{A}(X) \cup \bigcup_{i=1}^{k} \mathcal{A}(Y_i)$ berechnet.
- Für jedes Attribut $a \in \mathcal{A}_I(Y_j)$, $1 \leq j \leq k$, gibt es eine Regel, die den Wert von a in Abhängigkeit der Werte anderer Attribute $D(Y_j.a) \subseteq \mathcal{A}(X) \cup \bigcup_{i=1}^{k} \mathcal{A}(Y_i)$ berechnet.

Sprechweise: Das Attribut a ist *abhängig* vom Attribut b , falls $b \in D(X.a)$ bzw. $b \in D(Y_i.a)$.

Eine kontextfreie Grammatik mit einer Attributierung der terminalen und nichtterminalen Symbole und mit einer Zuordnung semantischer Regeln zu allen Produktionen heißt **attributierte Grammatik**.

Bemerkung: In der formalen Definition treten nur Funktionen im mathematischen Sinne auf, d.h. Funktionen, bei deren Auswertung keine Nebeneffekte auftreten.
Es werden später jedoch als semantische Regeln auch Prozeduraufrufe bzw. Code-Fragmente verwendet, die durchaus Nebeneffekte haben können, etwa Teile des Zwischencodes erzeugen und ausgeben.
Man kann durch Einführen von zusätzlichen „Dummy"-Attributen und Festlegung einer bestimmten Reihenfolge der Auswertung dies jedoch wieder auf die ursprüngliche Definition zurückführen.

Definition: Ein **attributierter Ableitungsbaum** eines Wortes $w \in L(G)$ ist ein Ableitungsbaum für w bezüglich der kontextfreien Grammatik G, bei dem in jedem Knoten für ein $X \in N \cup T$ auch die Attribute $\mathcal{A}(X)$ notiert sind.

Definition: Ein **ausgewerteter attributierter Ableitungsbaum** eines Wortes $w \in L(G)$ ist ein attributierter Ableitungsbaum für w, in dem jedes Attribut a eines mit $X \in (N \cup T)$ markierten Knotens einen Wert aus $\mathcal{W}(X.a)$ hat und die zugehörige semantische Regel für $X.a$ erfüllt (korrekt) ist (**annotierter Ableitungsbaum**).
Der Prozess des Zuordnens von Werten zu Attributen eines attributierten Ableitungsbaums heißt **Auswertung des attributierten Ableitungsbaums**.

Bem.:

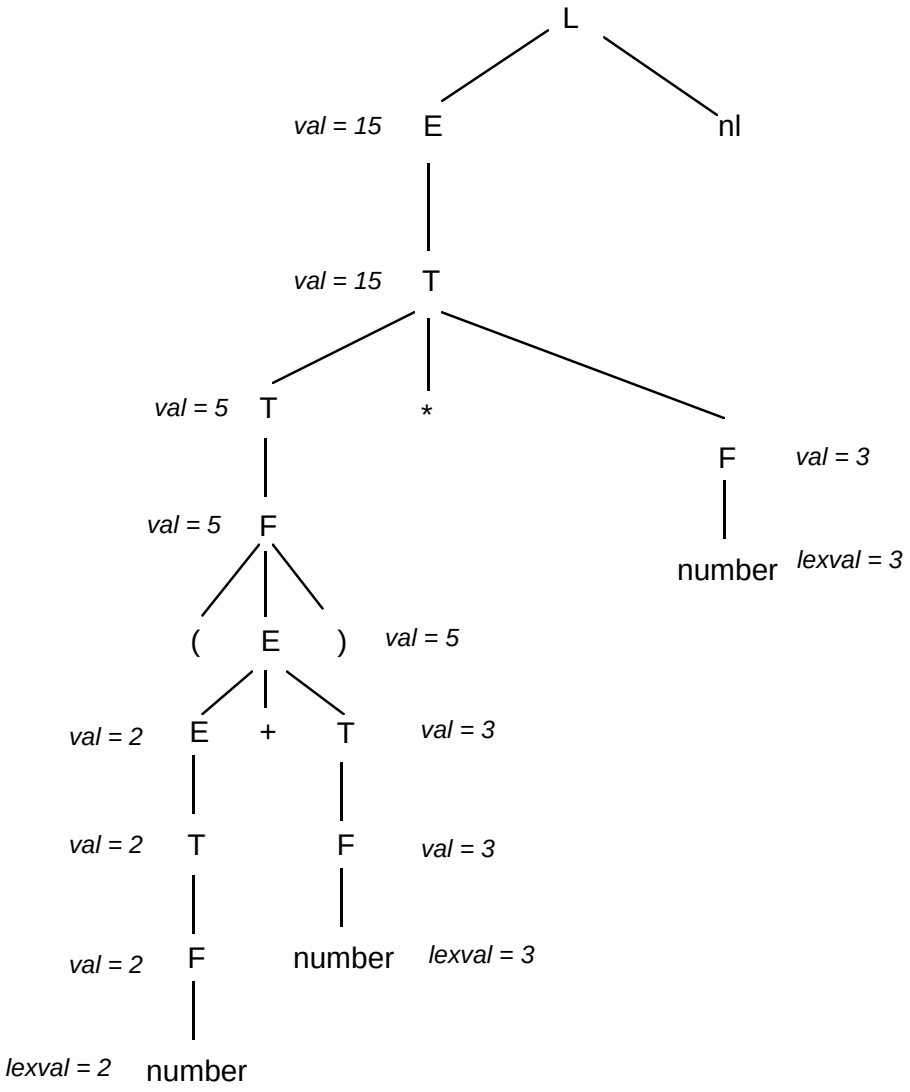
1) Nicht jeder attributierte Ableitungsbaum lässt sich auswerten!

2) Man beachte, dass bei der Verwendung von Funktionen mit Seiteneffekten, z.B. bei der Einführung globaler Variablen, Schwierigkeiten auftreten können und die Auswertung nicht mehr eindeutig ist. In diesem Fall ist die Auswertereihenfolge entscheidend!

3) Häufig wird ein synthetisches Attribut des Startsymbols der Grammatik ausgezeichnet. Dieses Attribut des Wurzelknotens des Ableitungsbaumes enthält dann nach der Auswertung „die Übersetzung des Wortes w".

Beispiel 2.1:
Gegeben ist eine kontextfreie Grammatik $G = (N, T', P, L)$ mit $N = \{L, F, E, T\}$,
$T' = \{\mathbf{nl}, +, *, (,), \mathbf{number}\}$ und $\mathcal{A}_S(E) = \mathcal{A}_S(F) = \mathcal{A}_S(T) = \{val\}$, $\mathcal{A}_S(\mathbf{number}) = lexval$.
Weiterhin sei $\mathcal{A}_S(L) = \mathcal{A}_I(L) = \mathcal{A}_I(E) = \mathcal{A}_I(F) = \mathcal{A}_I(T) = \mathcal{A}_I(\mathbf{number}) = \emptyset$.
Die folgende Attributierung bewirkt, dass ein Wort aus $L(G)$ als arithmetischer Ausdruck interpretiert und ausgewertet wird.
Die der Produktion $L \to E$ **nl** zugeordnete semantische Regel hat als Seiteneffekt das Ausdrucken des Wertes des arithmetischen Ausdrucks.

Produktionen	Semantische Regeln
$L \to E$ **nl**	print($E.val$)
$E \to E_1 + T$	$E.val := E_1.val + T.val$
$E \to T$	$E.val := T.val$
$T \to T_1 * F$	$T.val := T_1.val * F.val$
$T \to F$	$T.val := F.val$
$F \to (E)$	$F.val := E.val$
$F \to$ **number**	$F.val :=$ **number**$.lexval$



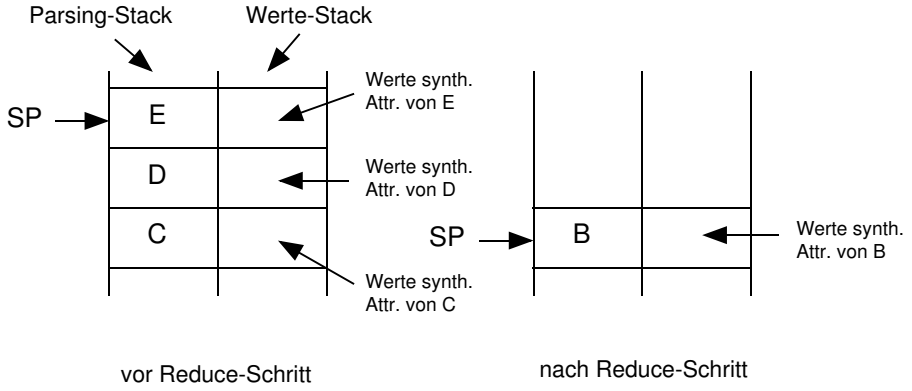
Ausgewerteter attributierter Ableitungsbaum für $w = (2 + 3) * 3$ **nl**.

Man beachte, dass unsere Beispielgrammatik nur synthetische Attribute enthält. Grammatiken dieses Typs haben einige Vorteile, so kann man zum Beispiel die Auswertung eines attributierten Ableitungsbaums bei diesen Grammatiken in einem Durchlauf durch den Baum „bottom-up“ durchführen.

Definition: Eine attributierte Grammatik, in der nur synthetische Attribute definiert sind, heißt **S-attributierte Grammatik**.

Die Auswertung attributierter Ableitungsbäume von S-attributierten Grammatiken kann parallel zum Parsing erfolgen, wenn ein bottom-up Parsingverfahren benutzt wird. Dazu müssen nur die Stack-Einträge im Parsing-Stack so groß gewählt werden, dass ausreichend Platz für die Werte der zugeordneten synthetischen Attribute vorhanden ist.
Vor einem Reduce-Schritt des Parsers befinden sich alle Symbole auf der rechten Seite einer Produktion samt den Werten ihrer Attribute auf dem Stack. Man kann also jetzt die der Produktion zugeordneten semantischen Regeln auswerten und die Werte der synthetischen Attribute des Symbols auf der linken Seite der Produktion berechnen. Die so berechneten Werte werden dann *nach* dem Reduce-Schritt dem obersten Stacksymbol zugeordnet.

Beispiel 2.2:
Sei $B \to CDE$ eine Produktion und seien für alle $a \in \mathcal{A}_S(B)$ Regeln zugeordnet. Die beiden Abbildungen symbolisieren die Situation vor und nach einem Reduktionsschritt des Parsers.



Für unsere Beispielgrammatik aus Beispiel 2.1 könnte man daher vielleicht wie folgt vorgehen: Da nur ein synthetisches Attribut für jedes nichtterminale Symbol der Grammatik existiert, ist der Zugriff auf die Werte des synthetischen Attributs recht einfach. $val(\mathrm{SP})$ soll z. B. den obersten Wert im Werte-Stack bezeichnen.
Der lexikale Scanner legt bei Erkennen einer Zahl den Tokenwert direkt in den Werte-Stack ab. SP' bezeichne die Position von SP nach einem Reduce-Schritt. Dann kann man die semantischen Regeln durch kurze Code-Fragmente angeben, die die semantischen Regeln realisieren.

Produktionen	Code-Fragmente
$L \to E$ **nl**	$\mathrm{print}(\mathrm{val}[\mathrm{SP}-1])$
$E \to E_1 + T$	$\mathrm{val}[\mathrm{SP}'] := \mathrm{val}[\mathrm{SP}-2] + \mathrm{val}[\mathrm{SP}]$
$E \to T$	
$T \to T_1 * F$	$\mathrm{val}[\mathrm{SP}'] := \mathrm{val}[\mathrm{SP}-2] * \mathrm{val}[\mathrm{SP}]$
$T \to F$	
$F \to (E)$	$\mathrm{val}[\mathrm{SP}'] := \mathrm{val}[\mathrm{SP}-1]$
$F \to$ **number**	

Abarbeitung des Wortes $(2+3)*3$ **nl**:

Eingabe	Pars.-Stack	Werte-Stack	angew. Prod
$(2+3)*3$ **nl**			
$2+3)*3$ **nl**	$($	$-$	
$+3)*3$ **nl**	$($**number**	-2	
$+3)*3$ **nl**	$(F$	-2	$F \to$ **number**
$+3)*3$ **nl**	$(T$	-2	$T \to F$
$+3)*3$ **nl**	$(E$	-2	$E \to T$
$3)*3$ **nl**	$(E+$	$-2-$	
$)*3$ **nl**	$(E+$**number**	$-2-3$	
$)*3$ **nl**	$(E+F$	$-2-3$	$F \to$ **number**
$)*3$ **nl**	$(E+T$	$-2-3$	$T \to F$
$)*3$ **nl**	$(E$	-5	$E \to E_1+T$
$*3$ **nl**	(E)	$-5-$	
$*3$ **nl**	F	5	$F \to (E)$
$*3$ **nl**	T	5	$T \to F$
3 **nl**	$T*$	$5-$	

usw.

Nun ein weiteres Beispiel, um zu zeigen, dass dieser Prozess des gleichzeitigen Parsens und Auswertens der Attribute keinesfalls immer so einfach funktioniert. Bei attributierten Grammatiken mit inheriten und synthetischen Attributen kann der Auswertungsprozess eventuell viel komplexer werden:

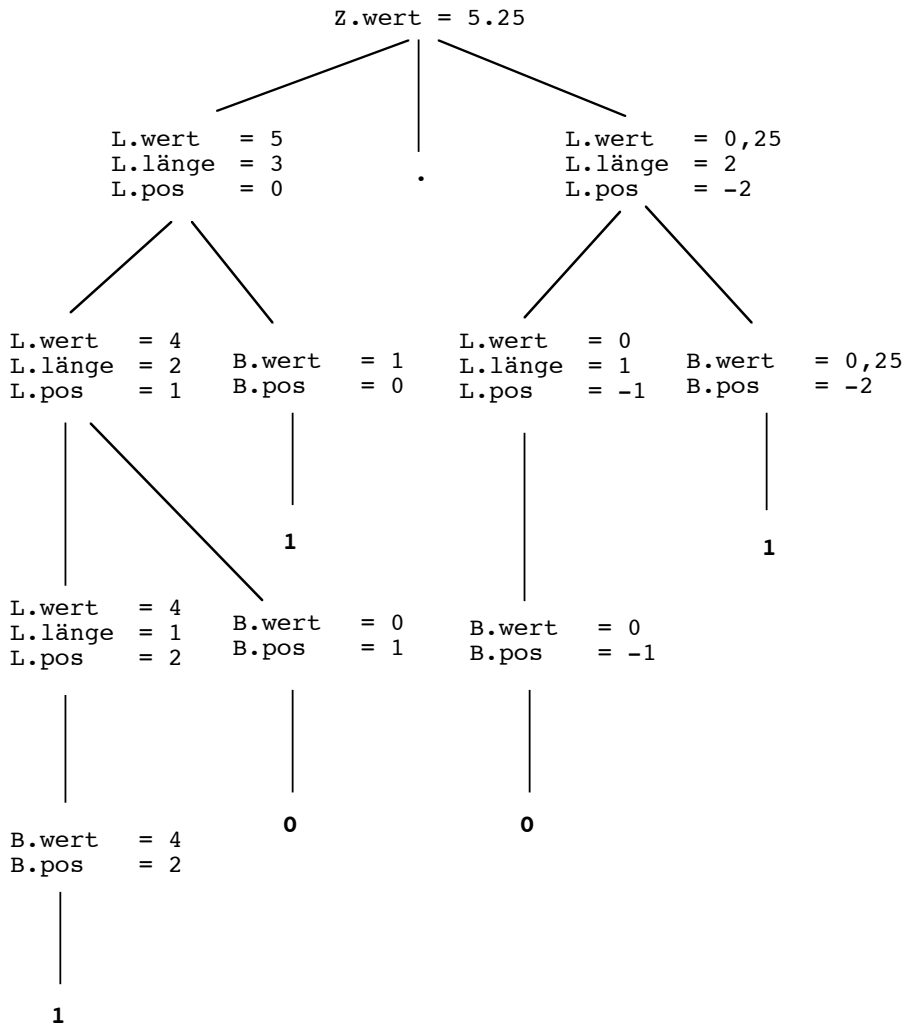
Beispiel 2.3:
Sei $G = (N, T, P, Z)$ eine kontextfreie Grammatik mit $N = \{Z, L, B\}$, $T = \{0, 1, .\}$ und

$$\begin{array}{ll} \mathcal{A}_S(L) = \{\mathit{länge}, \mathit{wert}\} & \mathcal{A}_I(L) = \{\mathit{pos}\} \\ \mathcal{A}_S(Z) = \{\mathit{wert}\} & \mathcal{A}_I(Z) = \emptyset \\ \mathcal{A}_S(B) = \{\mathit{wert}\} & \mathcal{A}_I(B) = \{\mathit{pos}\} \end{array}$$

Diese Grammatik erzeugt Wörter, die als Darstellung von Festkommazahlen im Dualsystem interpretiert werden können. Dabei steht das Symbol Z für eine Festkommazahl, L für eine Liste von 0 und 1 und B für ein Bit, also für 0 oder 1. Als Übersetzung eines Wortes soll der Wert der dargestellten Zahl berechnet und im synthetischen Attribut *wert* von Z abgelegt werden.

Produktion	**Semantische Regeln**
$Z \to L_1 \,.\, L_2$	$Z.\mathit{wert} := L_1.\mathit{wert} + L_2.\mathit{wert}$ $L_1.\mathit{pos} := 0$ $L_2.\mathit{pos} := -L_2.\mathit{länge}$
$L \to L_1 B$	$L.\mathit{länge} := L_1\mathit{länge} + 1$ $L.\mathit{wert} := L_1.\mathit{wert} + B.\mathit{wert}$ $L_1.\mathit{pos} := L.\mathit{pos} + 1$ $B.\mathit{pos} := L.\mathit{pos}$
$L \to B$	$L.\mathit{länge} := 1$ $L.\mathit{wert} := B.\mathit{wert}$ $B.\mathit{pos} := L.\mathit{pos}$
$B \to 0$	$B.\mathit{wert} := 0$
$B \to 1$	$B.\mathit{wert} := 2^{B.\mathit{pos}}$

Das Attribut *länge* gibt die Länge der Liste L an und *pos* die Position der Liste oder des Bits (entspricht dem Abstand vom „.“). Das Attributs *wert* repräsentiert den Beitrag zum Gesamtwert der Zahl.



Ausgewerteter attributierter Ableitungsbaum für $w = 101.01$.

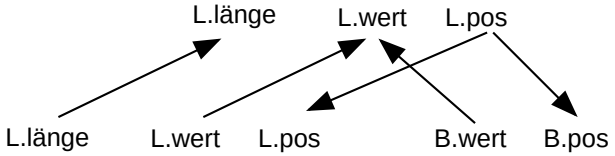
2.1.1 Konstruktion eines Abhängigkeitsgraphen für eine Produktion

Wie man am Beispiel 2.3 deutlich sehen kann, ist die Reihenfolge, in der die Attribute im Ableitungsbaum mit Werten versehen werden können, nicht ohne weiteres zu erkennen.
Um das Problem der Suche nach einer Auswertereihenfolge näher zu untersuchen, benötigt man das Konzept eines **Abhängigkeitsgraphen**.

Der Abhängigkeitsgraph $g(\pi)$ für die Produktion π wird wie folgt bestimmt:

```
for jedes Symbol X in π  do
    for jedes Attribut a von X do
        erzeuge einen neuen Knoten in g(π) mit Markierung X.a
for jede Regel ρ zu π do
    bestimmt ρ das Attribut X.a in Abhängigkeit der Attribute
    X_1.b_1, ... ,X_k.b_k, so erzeuge eine Kante in g(π) von jedem der
    Knoten X_i.b_i nach X.a
```

Beispiel 2.4:
Betrachtet man die Produktion $L \to LB$ aus dem Beispiel 2.3 oben, so erhält man den folgenden Abhängigkeitsgraph, wobei die Konvention so ist, dass die Knoten für die Attribute des Symbols auf der linken Seite „oben“ und die der Symbole auf der rechten Seite der Produktion in dieser Reihenfolge „unten“ angeordnet werden:



2.1.2 Konstruktion eines Abhängigkeitsgraphen für einen Ableitungsbaum

Der Abhängigkeitsgraph $G(\tau)$ für den Ableitungsbaum τ wird durch Aneinanderfügen von Abhängigkeitsgraphen der Produktionen schrittweise aufgebaut:

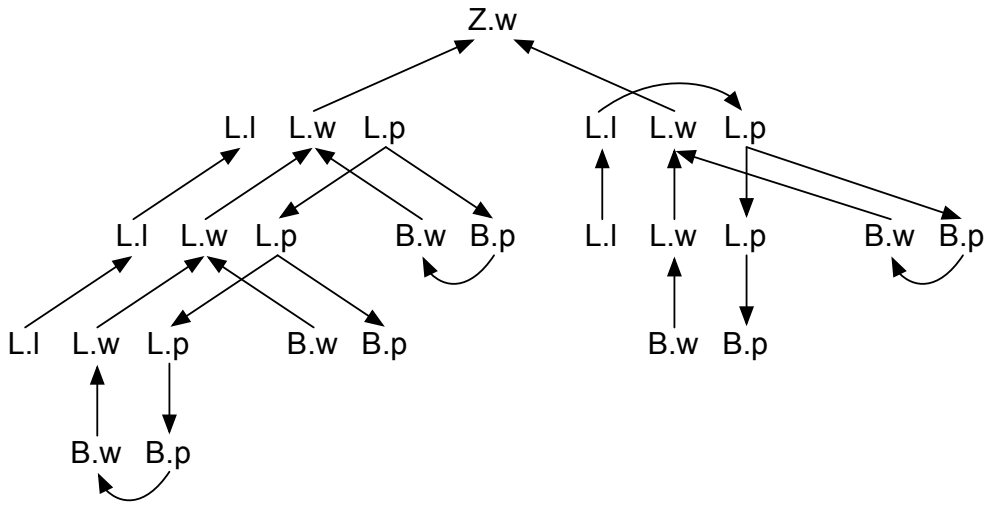
```
Erzeuge zunächst die Knoten S.a für jedes Attribut a des Startsymbols S.
Dann durchlaufe τ in Präordnung
    Sei n der nächste Knoten in Präordnung mit Markierung X in τ.
    Ist X ein nichtterminales Symbol, dann existieren in γ die
        Knoten X.b, b Attribut von X.
    Wurde auf X in τ die Produktion π angewendet,
        so füge g(π) zu G(τ) hinzu, wobei die Knoten X.b in G(τ) und g(π)
        identifiziert werden
```

Der Abhängigkeitsgraph eines Ableitungsbaumes kann nun benutzt werden, um eine **Auswertereihenfolge** für die Attribute im attributierten Ableitungsbaum festzulegen.
Es gilt:
Gibt es eine Kante von $X.a$ nach $Y.b$ im Abhängigkeitsgraphen, so muss das Attribut a von X *vor* dem Attribut b von Y mit einem Wert versehen werden.
Damit folgt sofort:

- Es existiert genau dann eine Auswertereihenfolge, wenn der Abhängigkeitsgraph des Ableitungsbaums azyklisch ist. (Stichwort: Topologisches Sortieren)

Hinweis: Es ist entscheidbar, ob eine gegebene attributierte Grammatik nur Worte erzeugen kann, deren attributierte Ableitungsbäume zyklenfreie Abhängigkeitsgraphen haben [34]. Dieser Test benötigt $O(e^n)$ Schritte, wobei n die „Größe“ der Grammatik beschreibt [33].

Beispiel 2.5:
So ergibt sich für das Beispiel 2.3 mit dem Wort $\omega = 101.01$ der folgende Abhängigkeitsgraph für den Ableitungsbaum:



Da dieser Ableitungsbaum offensichlicht azyklisch ist, existiert eine Auswertereihenfolge und der Baum kann ausgewertet werden.

2.2 Übersetzung eines Wortes w mit einer attributierten Grammatik

Um also eine durch eine attributierte Grammatik gegebene Übersetzung durchführen zu können, muss man die folgenden Schritte ausführen:

1) Parsen des vorgelegten Wortes w bzgl. der kontextfreien Grammatik und Konstruktion des Ableitungsbaums

2) Konstruktion des Abhängigkeitsgraphen und Festlegung einer Auswertereihenfolge

3) Auswertung der Attribute ergibt die Übersetzung als Wert eines ausgezeichneten synthetischen Attributs des Startsymbols.

Es gibt zwei wesentliche Methoden der Implementation einer derartigen allgemeinen Übersetzung:

1) **on-the-fly Methoden**, bei denen während des Parsings alle Attribute berechnet werden können. Dies erfordert signifikante Einschränkungen bzgl. der Attributierung wie etwa bei S-attributierten Grammatiken.

2) **tree-walk Methoden**, bei denen in einem oder mehreren Durchläufen durch den Ableitungsbaum oder einen daraus konstruierten Syntaxbaum die Attribute ausgewertet werden (*multi-visit attributierte Grammatiken*).

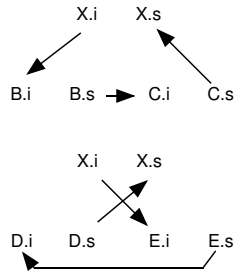
Den Fall 2 kann man noch unterteilen in

- **die Anzahl der Durchläufe ist unbegrenzt.** Jede nicht-zirkuläre attributierte Grammatik kann auf diese Weise ausgewertet werden. Man durchläuft den Baum wiederholt in einer vorbestimmten Ordnung, etwa in Präordnung. In jedem Knoten berechnet man die Attributwerte, die sich momentan aufgrund der gegebenen Abhängigkeiten und der bereits berechneten Werte bestimmen lassen.
- **die Anzahl der benötigten Durchläufe ist durch eine Konstante begrenzt.** (Diese Eigenschaft ist entscheidbar! [26])
  Weiterhin kann man noch bzgl. der Reihenfolge differenzieren, in der die Nachfolgerknoten eines Knotens im Ableitungsbaum ausgewertet werden, z.B.
  - immer links-nach-rechts
  - immer rechts-nach-links
  - alternierend bei jedem Durchlauf
  - in einer durch eine Permutation für jede Produktion festgelegten Reihenfolge.

  Weiterhin kann man unterscheiden, ob diese Reihenfolge statisch, also für alle Ableitungsbäume der Grammatik, oder dynamisch, je nach vorliegendem Ableitungsbaum, gewählt wird.

Beispiel 2.6:
Man betrachte die beiden Produktionen $X \to BC$ und $X \to DE$. Jedes Symbol habe ein inherites Attribut i und ein synthetisches Attribut s

Produktion	Regeln
$X \to BC$	$B.i := f_1(X.i)$
	$C.i := f_2(B.s)$
	$X.s := f_3(C.s)$
$X \to DE$	$E.i := g_1(X.i)$
	$D.i := g_2(E.s)$
	$X.s := g_3(D.s)$



Bei der ersten Produktion wäre eine links-nach-rechts Auswertung, bei der zweiten wäre eine rechts-nach-links Auswertung günstig. Können beide Produktionen in einem Ableitungsbaum beliebig oft untereinander auftreten, so ist die Zahl der notwendigen Durchläufe unbegrenzt, sofern eine feste Durchlaufordnung vorgegeben ist.
Andererseits könnte man mit einer durch eine Permutation für jede Produktion festgelegten Reihenfolge den Ableitungsbaum eventuell sogar in einem Durchlauf auswerten!

Eine weitere wichtige Teilklasse von attributierten Grammatiken, deren Ableitungsbäume sich in einem Durchlauf auswerten lassen, sind die L-attributierten Grammatiken [35].

Definition: Eine attributierte Grammatik $G = (N, T, P, S)$ heißt **L-attributiert**, wenn für jede Produktion $X \to Y_1 \dots Y_k$ in P, $X \in N$, $Y_i \in (N \cup T)$, $1 \le i \le k$, $k \ge 0$ und für jedes j, $1 \le j \le k$ gilt:

$$\text{Ist } a \in \mathcal{A}_I(Y_j), \text{ dann ist } a \text{ nur abhängig von Attributen aus } \mathcal{A}_I(X) \cup \bigcup_{i=1}^{j-1} \mathcal{A}(Y_i)$$

Bem.: Jede S-attributierte Grammatik ist auch L-attributiert.
Die attributierte Grammatik für die Dualzahlen aus dem Beispiel 2.3 ist nicht L-attributiert, da die Attributierung der Produktion $Z \to L_1 \,.\, L_2$ die Bedingungen verletzt.

Beispiel 2.7:
Die folgende L-attributierte Grammatik beschreibt beispielhaft Typ-Deklarationen in einem Programm.
Es sei $G = (N, T', P, D)$ mit $N = \{D, L, T\}$, $T = \{\textbf{int}, \textbf{float}, \textbf{id}, ',' \}$. Weiterhin gelte, dass L ein inherites Attribut *inh*, T ein vom Scanner gesetztes synthetisches Attribut *type* und **id** ein synthetisches Attribut *entry* besitzt. Alle anderen Symbole haben keine Attribute. Die hier verwendete Funktion *addType* fügt einen neuen Eintrag für eine Variable mit zugehörigem Typ in die Symboltabelle ein. Die Produktionen mit den semantischen Regeln sind:

Produktion	**Semantische Regeln**
$D \to T\ L$	$L.inh := T.type$
$T \to \textbf{int}$	$T.type =$ integer
$T \to \textbf{float}$	$T.type =$ float
$L \to L_1, \textbf{id}$	$L_1.inh = L.inh$
	$addType(\textbf{id}.entry, L.inh)$
$L \to \textbf{id}$	$addType(\textbf{id}.entry, L.inh)$

DIe Funktion *addType* ist natürlich eine Funktion mit Seiteneffekten. Da das Einsetzen eines Eintrags die anderen Einträge in der Symboltabelle nicht verändert, kann auf das Festlegen einer speziellen Auswertereihenfolge verzichtet werden.

Wir werden sehen, dass sich L-attributierte Grammatiken besonders in Verbindung mit Top-Down Parsern für die Übersetzung eignen, da in diesem Fall ebenfalls kein expliziter Ableitungsbaum benötigt wird.
Die Vorgehensweise wird klarer ersichtlich, wenn man den Zeitpunkt für die Auswertung der semantischen Regeln relativ zum Parsingprozess in einer gewissen Form festlegen kann. Dies ist besonders wichtig, wenn man semantische Regeln mit Seiteneffekten verwenden möchte.

Definition: Ein **syntax-gesteuertes Übersetzungsssschema (SDTS)** ist eine andere Darstellung einer attributierten Grammatik $G = (N, T, P, S)$. Die kontextfreien Produktionen $X \to Y_1 \dots Y_k$, $X \in N$, $Y_i \in N \cup T$, $1 \leq i \leq k$, $k \geq 0$ werden um die semantischen Regeln erweitert. Die Produktionen haben dann die Form $X \to \alpha_0 Y_1 \alpha_1 \dots \alpha_{k-1} Y_k \alpha_k$, wobei die α_i die Form $\alpha_i = \varepsilon$ oder $\alpha_i = \{$ semantische Regeln $\}$ haben. Die α_i heißen **Aktionen**.

Bedeutung: Durch die Einbettung der semantischen Regeln in die Produktion kann der Zeitpunkt der Ausführung der Regeln implizit festgelegt werden. Die Bedeutung dieser Notation sei dabei so, dass die semantischen Regeln in α_i ausgeführt werden, nachdem die Ableitung für Y_i erstellt wurde und bevor die Ableitung von Y_{i+1} beginnt. Da Aktionen auch auch Seiteneffekte, wie etwa eine `print`-Anweisung, enthalten können, ist die Reihenfolge der Ausführung der Aktionen wichtig.

Offensichtlich müssen die folgenden Regeln erfüllt sein, damit ein SDTS korrekt definiert ist:
Für alle $1 \leq i \leq k$ muß gelten:

1) Der Wert eines inheriten Attributs a von Y_i muß in einer der Aktionen $\alpha_0, \dots \alpha_{i-1}$ berechnet werden. Üblicherweise geschieht dies zum spätesten Zeitpunkt, also in α_{i-1}.

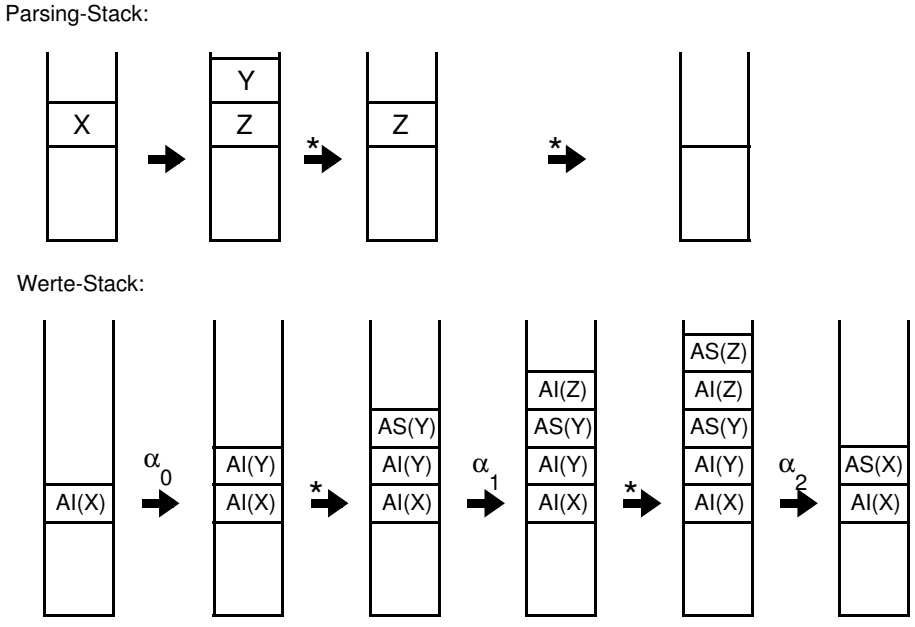
2) Der Wert eines synthetischen Attributs b von X kann erst berechnet werden, wenn alle zur Berechnung benötigten Werte bekannt sind. Üblicherweise geschieht dies auch zum spätesten Zeitpunkt, also in α_k.

Eine L-attributierte Grammatik erfüllt diese Bedingungen in jedem Fall.

2.3 Auswertung L-attributierter Grammatiken beim Top-Down Parsing

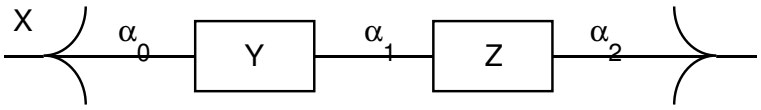
Wenn man eine LL-Grammatik mit einer L-Attributierung gegeben hat, kann man den Parsing- und den Auswerteprozess parallel durchführen.
Will man einen tabellengesteuerten Top-Down-Parser benutzen, reicht es im Gegensatz zu den S-attributierten Grammatiken nicht, einfach den Parsing-Stack zu erweitern.
Man benötigt zwar auch in diesem Fall einen zusätzlichen Attributwerte-Stack, der jedoch leider nicht genau synchron zum Parsing-Stack wächst und schrumpft. Man benötigt also zusätzlichen Aufwand zur Organisation des Werte-Stacks.

Beispiel 2.8:
Am Beispiel der Regel $X \to \alpha_0 Y \alpha_1 Z \alpha_2$ soll die Arbeitsweise erläutert werden.
In der Anfangssituation steht auf dem Parsing-Stack X und auf dem Werte-Stack steht ein Record, der die Werte der inheriten Attribute von X enthält. Die nächste anzuwendende Produktion sei $X \to YZ$.
Zunächst werden die inheriten Attribute von Y berechnet. Da man dazu die inheriten Attribute von X benötigt, müssen diese sich noch auf dem Werte-Stack befinden. Ist die Ableitung von Y beendet und auf dem Parsing-Stack wird Z das oberste Symbol, so müssen die berechneten synthetischen Attribute von Y ebenfalls auf dem Werte-Stack gelegt werden, da alle diese Werte eventuell zur Berechnung der inheriten Attribute von Z benötigt werden.
Wenn also ein Symbol vom Stack gelöscht wird, sollen auf dem Werte-Stack sowohl die inheriten als auch die synthetischen Attribute dieses Symbols stehen. Erst nach Berechnung der synthetischen Attribute von X können die Attributwerte von Y und Z vom Werte-Stack gelöscht werden.



Einfacher und natürlicher ist die Verbindung eines Parsers, der nach der Methode des rekursiven Abstiegs arbeitet, mit einer L-Attributierung. Als Beispielproduktion sei $X \to \alpha_0 Y \alpha_1 Z \alpha_2$ gewählt, wobei $X, Y, Z \in N$ gelte und $\alpha_0, \alpha_1, \alpha_2$ semantische Aktionen sind.
Es soll nun ein Parser konstruiert werden, der nach der Methode des rekursiven Abstiegs arbeitet. (**recursive descent parser**)
Wie üblich wird zunächst für jedes nichtterminale Symbol ein Syntaxgraph konstruiert. Man markiert dabei zusätzlich die Kanten des Graphen mit den Aktionen. Dann vereinfacht man die Syntaxgraphen.

Ausschnitt aus dem Syntaxgraph für X:



Für jeden Syntaxgraphen eines nichtterminalen Symbols X erzeugt man dann wie üblich eine Prozedur mit Namen `X`. Diese Prozedur bekommt für jedes inherite Attribut in $\mathcal{A}_I(X)$ einen Formalparameter und gibt die Werte der synthetischen Attribute in $\mathcal{A}_S(X)$ (z.B. als Record) zurück. Weiterhin hat die Prozedur `X` lokale Variablen für jedes synthetische Attribut von X und jeweils eine lokale Variable für jedes Attribut eines jeden im Syntaxgraphen auftretenden Symbols. Tritt ein Symbol mehrfach im Syntaxgraphen auf, so müssen entsprechend viele Exemplare der lokalen Variablen erzeugt werden.
Neben dem üblichen Code zur Steuerung der Ableitung beim rekursiven Abstieg werden

- bei terminalen Symbolen a die Werte der synthetischen Attribute von a in die lokalen Variablen gespeichert (als Initialisierung dieser Variablen)
- bei nichtterminalen Symbolen Y bekommen Prozeduraufrufe die Form
  $\texttt{syn}_1\texttt{, ..., syn}_r\texttt{ := Y(inh}_1\texttt{, ...,inh}_k\texttt{)}$,
  wobei $\texttt{inh}_1\texttt{, ..., inh}_k$ die lokalen Variablen für die inheriten und $\texttt{syn}_1\texttt{, ..., syn}_r$ für die synthetischen Attribute von Y sind.

  (Hier ist der Einfachheit halber eine Syntax angenommen worden, die es einer Prozedur erlaubt, mehrere Werte zurückzugeben.)
- die Aktionen durch entsprechende Code-Fragmente in die Parsing-Prozedur übertragen. In den semantischen Regeln auftretende Attribute werden dabei durch die korrespondierenden lokalen Variablen ersetzt.
- beim Verlassen der Prozedur die Werte der synthetischen Attribute zurückgegeben.

Beispiel 2.9:
Für die oben angegebene Beispielproduktion $X \to \alpha_0 Y \alpha_1 Z \alpha_2$ wird das folgende Code-Fragment in der Parsingprozedur für das Nichtterminale X erzeugt. `AI(X)` bezeichne dabei die inheriten, `AS(X)` die synthetischen und `W(AS(X))` die Wertebereiche der synthetischen Attribute von `X`.

```
procedure X (AI(X)) : W(AS(X))
var AS(X), AI(Y), AS(Y), AI(Z), AS(Z)
begin
    ...

    α'0   /* Codefragment zur Berechnung der AI(Y) gemäß α0 */
    AS(Y) := Y(AI(Y));    /* Aufruf von Y liefert Werte der AS(Y) */

    α'1   /* Codefragment zur Berechnung der AI(Z) gemäß α1 */
    AS(Z) := Z(AI(Z));    /* Aufruf von Z liefert Werte der AS(Z) */

    α'2   /* Codefragment zur Berechnung der AS(X) gemäß α2 */
    return AS(X)
    ...
end
```

2.4 Umformungen der Grammatiken

Manchmal ist es notwendig, Transformationen an den Produktionen der Grammatik vorzunehmen, damit ein deterministisches Parsen mit gleichzeitiger Auswertung der Atrribute möglich ist oder aber damit gewisse Regeln (mit Seiteneffekten) zur „richtigen“ Zeit durchgeführt werden. Welche Auswirkungen haben derartige Transformationen auf die Attributierung, oder wie muss man die Attributierung ändern, damit der Wert der Übersetzung gleich bleibt?
Hier sollen exemplarisch nur zwei Fälle betrachtet werden:

1) Das Entfernen von Aktionen, die *in* der rechten Seite von Produktionen eines bottom-up parsebaren SDTS auftreten, um ein deterministisches Bottom-Up-Parsing mit gleichzeitiger Atrributauswertung zu ermöglichen.

2) Das Entfernen linksrekursiver Produktionen bei S-attributierten Grammatiken, damit ein deterministisches Top-Down-Parsing möglich wird.

Betrachten wir zunächst den Fall 1).

Man ersetzt eine Aktion a, die nicht am Ende, sondern innerhalb einer Produktion ausgeführt werden muss, durch ein neues nichtterminales Symbol (Marker) und führt eine ε-Produktion für dieses Symbol mit Aktion a ein.

Beispiel 2.10:
Das folgende SDTS definiert eine Übersetzung einfacher arithmetischer Ausdrücke in Postfix-Notation, wobei die Übersetzung als Ausgabe der print-Anweisungen erscheint.

$$\begin{aligned} E &\rightarrow TR \\ R &\rightarrow +T\{\text{print}('+')\}R \mid -T\{\text{print}('-')\}R \mid \varepsilon \\ T &\rightarrow \textbf{number}\{\text{print}(\textbf{number}.\text{val})\} \end{aligned}$$

Zum Beispiel führt die Eingabe $3-2+1$ zur Ausgabe $3\,2-1+$. Wegen der Seiteneffekte der print-Anweisung müssen die semantischen Aktionen aber auch genau an diesen Stellen ausgeführt werden!
Eine Übertragung des obigen SDTS in eine S-attributierte Grammatik, die eine entsprechende Übersetzung parallel zum Parsingprozess ermöglicht, erfordert die Einführung der beiden neuen nichtterminalen Symbole M und N mit jeweiligen ε-Produktionen:

$$\begin{aligned} E &\rightarrow TR \\ R &\rightarrow +TMR \mid -TNR \mid \varepsilon \\ T &\rightarrow \textbf{number}\{\text{print}(\textbf{number}.\text{val})\} \\ M &\rightarrow \varepsilon\{\text{print}('+')\} \\ N &\rightarrow \varepsilon\{\text{print}('-')\} \end{aligned}$$

Die zusätzlich eingeführten Symbole und Produktionen erlauben weiterhin ein deterministisches bottom-up Parsen, wobei die semantichen Aktionen jetzt wie üblich bei jedem Reduktionsschritt ausgeführt werden.

Im Fall 2) kann man die Linksrekursionen mit dem „üblichen“ Verfahren entfernen, muss aber die Attributierung ändern:

Seien $A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m$ alle A-Produktionen, deren rechte Seite mit A beginnt und seien $A \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$ alle restlichen A-Produktionen. Weiterhin gelte $\alpha_i \neq \varepsilon$ für $1 \leq i \leq m$. Man ersetzt diese Produktionen durch

$$A \to \beta_1 A', \ldots, A \to \beta_n A' \text{ und } A' \to \alpha_1 A', \ldots, A' \to \alpha_m A', A' \to \varepsilon$$

wobei A' ein neues nichtterminales Symbol ist.
Man beachte, dass durch diese Änderung an der Grammatik andere Ableitungsbäume entstehen, in denen die synthetischen Attribute nicht mehr so wie in der ursprünglichen Grammatik berechnet werden können, da die benötigten Werte „an falscher Stelle im Baum“ stehen.
In der transformierten Grammatik bekommt das neu eingeführte Symbol A' deshalb ein inherites und ein synthetisches Attribut. Das neu eingeführte inherite Attribut benötigt man, um die Werte synthetischer Attribute in dem jetzt veränderten Ableitungsbaum nach „unten“ reichen zu können, damit sie als Argumente für die entsprechenden semantischen Regeln zur Verfügung stehen. Das synthetische Attribut wird benötigt, um den berechneten Wert wieder nach oben zum nichtterminalen Symbol A zu transportieren. Die semantischen Aktionen müssen natürlich ebenfalls angepasst werden.
Diese Transformation soll hier nur beispielhaft an einer S-attributierten Grammatik vorgestellt werden:

Beispiel 2.11:
Die folgende links-rekursive, S-attributierte Grammatik sei gegeben. Die auftretenden nichtterminalen Symbole haben wie das Token **number** alle nur das synthetische Attribut *val*.

Produktion	sem. Regel
$E \to E_1 + T$	$E.val := E_1.val + T.val$
$E \to E_1 - T$	$E.val := E_1.val - T.val$
$E \to T$	$E.val := T.val$
$T \to (E)$	$T.val := E.val$
$T \to \mathbf{number}$	$T.val := \mathbf{number}.val$

Entfernt man die Links-Rekursionen und ändert die Attributierung entsprechend, so erhält man eine rechts-rekursive Grammatik. Das neu hinzugekommene nichtterminale Symbol R bekommt ein inherites Attribut i und ein synthetisches Attribut s. Insgesamt erhält man eine äquivalente L-attributierte Grammtik, die in Form eines SDTS notiert wird.

$$\begin{aligned}
E &\to T\ \{R.i := T.val\}\ R\ \{E.val := R.s\} \\
R &\to +T\ \{R_1.i := R.i + T.val\}\ R_1\ \{R.s := R_1.s\} \\
R &\to -T\ \{R_1.i := R.i - T.val\}\ R_1\ \{R.s := R_1.s\} \\
R &\to \varepsilon\ \{R.s := R.i\} \\
T &\to (E\ \{T.val := E.val\}\ ) \\
T &\to \mathbf{number}\ \{T.val := \mathbf{number}.val\}
\end{aligned}$$

Konstruiert man jetzt für die obige L-attributierte Grammatik einen recursive-descent-Parser wie im Abschnitt 2.3 beschriebenn, so erhält man die folgenden Prozeduren:

```
procedure T(): integer;
Tval, Eval, numberval: integer;
begin
     if token = "("  then
         begin
         nextToken();
         Eval := E();
         Tval := Eval;
         if not (token = ")" ) then
              error();
         else
              begin
              nextToken();
              return(Tval);
              end;
         end;
     else if token = "number" then
         begin
         numberval := lexval;
         nextToken();
         Tval := numberval;
         return(Tval);
         end;
     else error();
```

```
procedure E(): integer;
Eval, Tval, Rs, Ri : integer;
begin
     Tval := T();
     Ri := Tval;
     Rs := R(Ri);
     Eval := Rs;
     return(Eval);
end;
```

```
procedure R(Ri: integer) : integer;
Rs, Tval, R1s, R1i : integer;
begin
     if token = "+" then
         begin             /* Aufruf des lexikalen Scanners, */
         nextToken();      /* setzt die globale Variable token */
         Tval := T();
         R1i := Ri + Tval;
         R1s:= R(R1i);
         Rs := R1s;
         return(Rs);
         end;
     else if token = "-" then
         begin             /* Aufruf des lexikalen Scanners */
         nextToken();      /* setzt die globale Variable token */
         Tval := T();
         R1i := Ri - Tval;
         R1s:= R(R1i);
         Rs := R1s;
         return(Rs);
         end;
     else if token = ")" or token = "$" then   /* Symbole in FOLLOW(R) */
         begin
         Rs := Ri;
         return(Rs);
         end;
     else error();
end;
```