SYSTEMATISCHE ANALYSE SEMANTISCHER ABHAENGIGKEITEN

Uwe Kastens
Fakultaet fuer Informatik der Universitaet Karlsruhe

1. Ueberblick

Die statische Semantik einer Programmiersprache kann beschrieben werden, indem man den Sprachkomponenten Attribute zuordnet und Abhaengigkeiten zwischen den Attributen angibt. In diesem Bericht wird beschrieben, wie solche Zusammenhaenge systematisch untersucht werden koennen. Es wird ein Verfahren angegeben, mit dem aus der syntaktischen und semantischen Definition einer Sprache eine Vorschrift ermittelt wird, nach der die semantischen Eigenschaften und Zusammenhaenge jedes Satzes der Sprache ermittelt werden koennen. Der Algorithmus kann zusammen mit einem Algorithmus zur Generierung eines Zerteilers den Kern eines Uebersetzer-erzeugenden Systems bilden. Mit solch einem System koennen die Uebersetzer-Prozeduren, die die semantische Analyse durchfuehren, automatisch aus der Sprachdefinition generiert werden. Die Komplexitaet der ausdrueckbaren semantischen Zusammenhaenge wird durch das Verfahren nicht beschraenkt.

Dieses Verfahren unterscheidet sich von den bisher bekannten dadurch, dass zur Analyse der semantischen Zusammenhaenge eine andere Methode angewandt wird als zur Ermittlung der syntaktischen Struktur. Solch ein Vorgehen erscheint uns zweckmaessig, da die Algorithmen zur syntaktischen und semantischen Analyse unterschiedliche Anforderungen hinsichtlich des zu betrachtenden Kontextes stellen. Systematische Verfahren zur syntaktischen Analyse sind im allgemeinen so konzipiert, dass die Struktur eines Satzes der Sprache in einem einzigen Durchgang unter Betrachtung eines moeglichst geringen Kontextes ermittelt werden kann (z.B. LR- oder LL-Verfahren). Die Analyse der Semantik vieler Programmiersprachen erfordert jedoch die Betrachtung eines groesseren Kontextes und in den meisten Faellen mehrere Durchgaenge durch den syntaktisch analysierten Satz der Sprache. Sie kann deshalb - zumindest nicht vollstaendig - in das Verfahren zur syntaktischen Analyse integriert werden.

Unser Verfahren basiert auf dem Ansatz von Knuth in [Kn68]. Dort werden semantische Eigenschaften von Sprachkomponenten durch Zuordnen von Attributen beschrieben und die Abhaengigkeiten zwischen den Attribut-Werten durch Graphen dargestellt. Knuth zeigt, wie aus der Untersuchung dieser Graphen festgestellt werden kann, ob die semantischen Definitionen der Sprache so beschaffen sind, dass zu jedem Satz der Sprache alle Attribut-Werte ermittelt werden koennen. Eine Sprache mit dieser Eigenschaft nennt er dann "semantisch wohldefiniert". Diese Untersuchungen wurden in modifizierter Form in unser Verfahren aufgenommen.

Wir gehen davon aus, dass die abstrakte syntaktische Struktur eines Satzes zu einer gegebenen Grammatik durch einen Ableitungs-Baum (AB) beschrieben wird. Jeder Knoten in einem AB repraesentiert ein Symbol des Vokabulars der Grammatik. Alle Knoten, die das gleiche Symbol repraesentieren, nennen wir der gleichen "Knoten-Klasse" zugehoerig. Alle Knoten, die zusammen mit ihren direkten Nachfolge-Knoten eine Anwendung der gleichen Ableitungsregel repraesentieren, nennen wir vom gleichen "Knoten-Typ".

Die semantischen Eigenschaften von Sprachkomponenten werden durch Attribut-Werte beschrieben, die den Knoten des AB zugeordnet sind. Allen Knoten einer Klasse werden die gleichen Attribute zugeordnet. Sie koennen abhaengig vom Kontext, in dem der jeweilige Knoten steht, verschiedene Werte annehmen. Den so erweiterten AB nennen wir einen "attributierten Ableitungs-Baum (AAB)".

Zu jeder syntaktischen Regel der Grammatik werden Vorschriften angegeben, nach denen die Berechnung der Werte bestimmter Attribute von Nichtterminalen erfolgt, die in dieser Regel auftreten. In den Berechnungsvorschriften koennen die Werte anderer Attribute verwendet werden.

Um die Werte aller Attribute in einem AAB zu berechnen, muss man im allgemeinen den AAB oder einige seiner Unterbaeume mehrfach in unterschiedlicher Richtung durchlaufen. Unser Verfahren ordnet jedem Knoten-Typ p (d.h. jeder Regel p) eine Sequenz von "Besuchen" zu, die beschreibt, in welcher Reihenfolge beim Durchlauf durch den AAB von einem Knoten des Typs p zu den Knoten in seiner direkten Nachbarschaft uebergangen werden muss. (Dabei koennen Knoten mehrfach besucht werden.) Ausserdem wird ermittelt, welche Attribut-Werte zwischen zwei solchen Besuchen berechnet werden koennen.

Diese Besuchssequenzen legen eindeutig fest, wie jeder zulaessige AAB zu einer bestimmten Sprache durchlaufen werden kann, damit die Werte aller Attribute berechnet werden koennen. Aus den Besuchssequenzen koennen fuer einen Uebersetzer die Programmstrukturen (z.B. rekursive Prozeduren oder Koroutinen) konstruiert werden, mit denen der AAB durchlaufen und die Attribute berechnet werden.

In Abschnitt 12 wird die Anwendung des Verfahrens am Beispiel einer einfachen Programmiersprache demonstriert.

[Kn68] Knuth, D.E., Semantics of Context-Free Languages, Mathematical Systems Theory, vol. 2, pp. 127-145, 1968

2. Notation

Die hier angegebenen Schreibweisen und Abkuerzungen werden in den folgenden Abschnitten nur mit der hier festgelegten Bedeutung verwendet. (Indizes werden in eckigen Klammern notiert; z.B. X[i], b[i,l].)

G=(V,T,P,S)	(kontext-freie) Grammatik
V	Vokabular
T∈V	Terminal-Symbole
P	Menge der syntaktischen Regeln
S∈V\T	Zielsymbol
U,W,X,Y,Z ∈ V\T	beliebige Nichtterminale
u,v,w,x,y ∈ V*	beliebige Zeichenfolgen
p,q,r,s,t ∈ P	syntaktische Regeln
p:Y->X[1]....X[n]	Notation einer Regel p. Dabei werden Terminal-Symbole auf der rechten Seite nicht angegeben, da sie in diesem Zusammenhang keine Bedeutung haben.
M,N	Mengen von Attributen
A(X)	Menge der Attribute von X
A[der](X)	Menge der abgeleiteten (derived) Attribute von X
A[in](X)	Menge der erworbenen (inherited) Attribute von X
a∈A(X)	Attribut
a=g[p](a[1],...,a[n])	semantische Vorschrift zur Berechnung des Wertes von a, die der Regel p zugeordnet ist
C[p]	Graph zur Beschreibung der durch alle g[p] gegebenen Abhaengigkeiten
D[p]	durch Huellenbildung aus C[p] gewonnener Graph
E[p]	durch Huellenbildung aus D[p] gewonnener Graph (Falls die Zuordnung zu einer Regel p nicht wesentlich ist, wird auch C,D,E geschrieben.)
k=(a[1],a[2])	Kante in einem Graphen von a[1] nach a[2]
w[p](a)=(up,down)	Wertepaar, das dem Attribut a in einem E[p] zugeordnet ist.
<	lineare Ordnungsrelation ueber den Wertepaaren (up, down)
b[i,l]	l-ter Besuch eines Knotens X[i]
F[p]	Besuchs-Sequenz (Folge von Attributen und Besuchen)

3. Voraussetzungen zur Sprachdefinition

Fuer unser Verfahren wird vorausgesetzt, dass die Grammatik G kontext-frei ist und keine unnoetigen Regeln enthaelt. Fuer jedes Y∈V\T gilt also: Es gibt eine Zeichenfolge uYv, die aus S ableitbar

ist, und jedes w in einer Regel p:Y→w gilt: w ist zu einer terminalen Zeichenfolge ableitbar.

Die semantischen Eigenschaften und Zusammenhaenge der Sprache werden durch Attribute formuliert. Jedem Nichtterminal X ist eine Menge von Attributen A(X) zugeordnet. Ein Attribut kann unterschiedliche Werte annehmen, abhaengig vom Kontext, in dem das Nichtterminal steht und von den Attribut-Werten anderer Nichtterminale in diesem Kontext. Die Vorschriften (semantische Regeln), nach denen die Werte der Attribute zu ermitteln sind, werden den syntaktischen Regeln zugeordnet. Wir nennen a∈A(X) ein abgeleitetes (derived) Attribut von X, wenn es zu einer Regel p:X→v eine Berechnungsvorschrift fuer a gibt. a∈A(X) heisst erworbenes (inherited) Attribut von x, wenn es zu einer Regel p:Y→uXv eine Berechnungsvorschrift fuer a gibt.

In den weiteren Untersuchungen wird vorausgesetzt, dass fuer die Attribute und Berechnungsvorschriften folgende Aussagen gelten:

a) Kein Attribut kann zugleich abgeleitet und erworben sein, d.h. fuer jedes X∈V\T muss gelten
   $A[der](X) \vee A[in](X) = A(X)$ und
   $A[der](X) \wedge A[in](X) = \emptyset$.
b) Fuer jedes abgeleitete (bzw.erworbene) Attribut von X muss es zu jeder Regel p:X→v (bzw. p:Y→uXv) eine Berechnungsvorschrift geben.
c) Fuer das Zielsymbol S gilt $A[in](S) = \emptyset$.

Fuer die Untersuchungen in den Abschnitten 7 bis 10 wird ausserdem die "semantische Wohldefiniertheit" der Sprache vorausgesetzt. In Abschnitt 6 wird diese Eigenschaft definiert und ein Verfahren zu ihrer Pruefung angegeben.

4. Strategie des Baum-Durchlaufes

Es ist das Ziel unseres Verfahrens, aus den semantischen Abhaengigkeiten Vorschriften zu ermitteln, die angeben, wie jeder AB zu durchlaufen ist, damit alle Attribut-Werte aller Knoten bestimmt werden koennen. Nimmt man an, dass der AB vollstaendig aufgebaut ist, bevor die Attribut-Werte berechnet werden, so kann man zu jeder Regel p:Y→X[1]...X[n] eine "Besuchssequenz" angeben, die die Wege durch jeden Knoten der Klasse Y und des Typs p und die bei jedem Besuch des Knotens berechenbaren Attribute bestimmt. Haengt die Berechnung von Attribut-Werten ab, deren Berechnungsvorschriften nicht p zugeordnet sind (a∈A[in](Y)vA[der](X[i])), so wird der entsprechende, benachbarte Knoten besucht (X[i] oder der Vorgaenger von Y).

Unser Verfahren erlaubt es darueber hinaus, die Besuchssequenzen so zu bestimmen, dass schon beim Aufbau des AB Attribute berechnet werden

koennen. in diesem Fall bestimmt die Ordnung, in der der Baum aufgebaut wird, die Reihenfolge der Besuche im ersten Durchlauf. Derartige Annahmen wirken sich nur auf den in Abschnitt 8 (Bewertung der Abhaengigkeits-Graphen) beschriebenen Teil unseres Verfahrens aus. Wir nehmen dort an, dass der AB von den Endknoten zur Wurzel hin aufgebaut wird (bottom-up) und dass dabei schon Attribut-Werte bestimmt werden, soweit es die Abhaengigkeiten zulassen. In Abschnitt 11 wird angegeben, wie das Verfahren zu modifizieren ist, falls eine andere Durchlaufstrategie vorgegeben wird (z.B. top-down-Aufbau des AAB, oder Ermittlung der Attribut-Werte nach dem vollstaendigen Aufbau des AA).

5. Beschreibung semantischer Abhaengigkeiten durch Graphen

Zu jeder Regel p:Y->X[1]...X[n] sind durch die Berechnungsvorschriften fuer die Attribut-Werte Abhaengigkeiten zwischen den Attributen von Y,X[1],...,X[n] definiert.

Es wird nun zu jeder Regel p ein gerichteter Graph C[p] konstruiert, der diese Abhaengigkeiten beschreibt.

Die Knoten von C[p] sind alle Attribute a$\in$ A(Y) v A(X[1]) v...v A(X[n]). In C[p] fuehrt genau dann eine Kante k=(a[i],a) von a[i] nach a, wenn es zu p eine Berechnungsvorschrift fuer a gibt, in der der Wert von a von dem Wert von a[i] abhaengt: a=g[p](a[1],...,a[i],... a[m]).

Wegen der Voraussetzung b in Abschnitt 3 hat jeder Graph C die Eigenschaften

a) Nur fuer a$\in$ A[der](Y) v A[in](X[1]) v...v A[in](X[n]) gibt es zu der Regel p Berechnungsvorschriften g[p]. In C[p] muenden deshalb nur in diese Knoten Kanten.
b) Ist in einer Berechnungsvorschrift a=g[p] g[p] eine konstante Funktion (d.h. der Wert von a wird unabhaengig von anderen Attribut-Werten berechnet), so muenden in C[p] keine Kanten in a.

In den folgenden Abschnitten werden wir die Graphen C so vervollstaendigen, dass sie auch alle Abhaengigkeiten zwischen Attributen beschreiben, die durch jede moegliche Zusammensetzung solcher Graphen zu einem AAB verursacht werden koennen. Dieser Vorgang ist als Huellenbildung ueber den Graphen bezueglich der Kontext-Abhaengigkeiten zu verstehen.

6. Semantische Wohldefiniertheit

Wir wollen nach [Kn68] eine Grammatik "semantisch wohldefiniert" nennen, wenn zu ihr kein AAB konstruiert werden kann, in dem einige Attribute zyklisch voneinander abhaengen, und ihre Werte deshalb nicht berechnet werden koennen.

Diese Eigenschaft der Grammatik kann mit Hilfe der Graphen C ueberprueft werden. Dazu vervollstaendigen wir jeden Graphen C[p] zu einer Regel p:Y->X[1]...X[n], so dass er auch alle Abhaengigkeiten zwischen den Attributen von Y, X[1],...,X[n] beschreibt, die durch jede moegliche Ableitung der X[i] gegeben sind (Regeln q:X[i]->u).

Zu jedem Graphen C[p] wird ein Graph D[p] konstruiert. Die Knotenmengen von C[p] und D[p] sind gleich. D[p] enthaelt eine Kante k=(a[1],a[2]) von a[1] nach a[2] genau dann, wenn k in C[p] enthalten ist oder wenn gilt a[1]∈A(X[i]) und a[2]∈A(der)(X[i]), $1 \leq i \leq n$ und es gibt einen Graphen D[q] zu einer Regel q:X[i]->u, in dem ein Weg von a[1] nach a[2] fuehrt. Die in D[p] aber nicht in C[p] enthaltenen Kanten nennen wir "durch die Regeln q in D[p] induziert".

Sind alle Graphen D[p] zyklenfrei, so kann man keinen AAB konstruieren, der einen Zyklus enthaelt - die Grammatik ist semantisch wohldefiniert.

Enthaelt ein Graph D[p] zu einer Regel p:Y->X[1]...X[n] einen Zyklus, so sind folgende Faelle zu unterscheiden:

a) Auf dem Zyklus liegen mindestens zwei Kanten, die durch verschiedene Regeln q:X[i]->u und r:X[i]->v in D[p] induziert sind. In diesem Fall fuehrt weder die Ableitungssequenz
Y=>X[1]...X[i]...X[n]=>X[1]...u...X[n] noch
Y=>X[1]...X[i]...X[n]=>X[1]...v...X[n]
zu einem Zyklus im AAB. Falls keine weiteren Zyklen auftreten, ist die Grammatik semantisch wohldefiniert. Der Zyklus in D[p] kann durch eine Transformation der Grammatik beseitigt werden: Man ersetzt die Regel p durch zwei Regeln p[1]:Y->X[1]...u...X[n] und p[2]:Y->X[1]...v...X[n].

b) In allen anderen Faellen laesst sich ein AAB konstruieren, der einen Zyklus enthaelt. Die Grammatik ist dann nicht semantisch wohldefiniert.

Alle weiteren Untersuchungen setzen voraus, dass alle Graphen D[p] zyklenfrei sind.

7. Erweiterung der Abhaengigkeits-Graphen

Die in Abschnitt 6 konstruierten Graphen D werden durch weitere Huellenbildung zu Graphen E vervollstaendigt.

Die Knotenmengen jedes D[p] und E[p] sind gleich. Ein Graph E[p] zu einer Regel p:Y->X[1]...X[n] enthaelt genau dann eine Kante k=(a[1],a[2]), wenn eine der folgenden Bedingungen erfuellt ist:

a) k ist in D[p] enthalten.
b) a[1]∈A(Y) und a[2]∈A[in](Y) und es gibt einen Graphen E[q] zu einer Regel q:Z->uYv, in dem ein Weg von a[1] nach a[2] fuehrt. Diese Kanten beschreiben die Abhaengigkeiten zwischen Attributen von Y, die durch jede moegliche Einbettung von Y in einen Kontext verursacht werden.
c) a[1]∈A(Y) und a[2]∈A[der](Y) und es gibt einen Graphen D[r] zu einer Regel r:Y->w, der eine Kante von a[1] nach a[2] enthaelt.
d) a[1]∈A(X[i]) und a[2]∈A[in](X[i]) und es gibt einen Graphen E[s] zu einer Regel s:U->xX[i]y, der eine Kante von a[1] nach a[2] enthaelt.

Die in c und d eingefuegten Kanten beschreiben Abhaengigkeiten zwischen Attributen, die bestehen, falls Y bzw. X[i] in einem anderen als durch die Regel p gegebenen Kontext stehen. (Diese Kanten gewaehrleisten, dass die Bewertung der Graphen und die Ermittlung der Besuchssequenzen zunaechst fuer jeden Graphen E[p] unabhaengig durchgefuehrt werden kann.)

Wir nennen die in E[p] aber nicht in D[p] enthaltenen Kanten "durch die Regeln q in E[p] induziert".

Es ist moeglich, dass die so konstruierten Graphen E Zyklen enthalten. In diesem Fall kann man jedoch die Grammatik so transformieren, dass ein solcher Graph durch mehrere zyklenfreie Grapnen ersetzt wird. Da alle Graphen D zyklenfrei sind, muss ein Zyklus in einem Graphen E[p] mindestens zwei Kanten enthalten, die durch verschiedene Regeln q in E[p] induziert sind. Setzt man nun die Regel q in p ein und/oder fuehrt neue Nichtterminale fuer ein X[i] mit entsprechenden neuen Regeln ein, so enthalten die Graphen E den urspruenglichen Zyklus nicht mehr.

Bei den folgenden Untersuchungen setzen wir voraus, dass alle Graphen E zyklenfrei sind.

8. Bewertung der Abhaengigkeits-Graphen

In diesem Abschnitt werden den Attributen a in den Graphen E[p] Wertepaare (up,down)=w[p](a) zugeordnet. Sei p:Y->X[1]...X[n], dann bedeutet w[p](a)=(m,n):
a kann berechnet werden, bevor von Y aus zum m+1-ten mal der Vorgaenger-Knoten besucht wird und bevor zwischen zwei Besuchen des Vorgaenger-Knotens (bzw. vor dem 1. Besuch falls m=0) zum n+1-ten mal ein Knoten X[i] besucht wird.

Zwei Wertepaare w(a[1])=(m[1],n[1]) und w(a[2])=(m[2],n[2]) sind gleich, wenn sie komponentenweise gleich sind.

Es wird eine Relation $<$ wie folgt definiert:
w(a[1])=(m[1],n[1])$<$w(a[2])=(m[2],n[2])
genau dann,wenn
m[1]$<$m[2] oder (m[1]=m[2] und n[1]$<$n[2]).

Die Bewertung der Attribute erfolgt so, dass a[1] vor a[2] berechnet werden kann, wenn w(a[1])$<$w(a[2]), und a[1] und a[2] beim gleichen Besuch berechnet werden koennen, wenn gilt w(a[1])=w(a[2]).

Zunaechst wird in einem Graphen E[p] zu einer Regel p:Y->X[1]...X[n] denjenigen Attributen, deren Wert beim ersten Besuch des Knotens Y berechnet werden kann, das Wertepaar (0,0) zugeordnet.

Dies sind alle Attribute a$\in$A[der](Y)vA[in](X[i]), $1 \leq i \leq n$, zu denen es eine konstante Berechnungsvorschrift a=g[p] gibt.

Aufgrund der gewaehlten Strategie fuer den Durchlauf durch den AAB kann in E[p] auch allen Attributen von X[i], die ohne einen Besuch des Vorgaenger-Knotens berechnet werden koennen, das Wertepaar (0,0) zugeordnet werden. Dies sind die Attribute a$\in$A[der](X[i]), in die in E[p] keine Kanten muenden.

Wir nehmen zunaechst an, dass alle Attribute a$\in$A[in](Y), in die keine Kanten muenden, nach dem ersten Besuch des Vorgaenger-Knotens berechnet sind, und ordnen ihnen das Wertepaar (1,0) zu. Nach der Bewertung aller Graphen wird diese Annahme in Abschnitt 10 ueberprueft und - falls erforderlich - korrigiert.

Alle uebrigen Attribute in E[p] werden nun sukzessive wie folgt bewertet. Sei a ein noch nicht bewertetes Attribut in E[p], k[j]=(a[j],a) alle Kanten, die in a muenden und alle a[j] seien bewertet. Dann kann a ein Wertepaar zugeordnet werden. Es sind dabei folgende Faelle zu unterscheiden:

a) Falls gilt a$\in$A[der](Y) v A[in](X[i]), $1 \leq i \leq n$ (es gibt dann eine Berechnungsvorschrift a=g[p](....)), dann ist w(a)=max(w(a[j])). Die Maximum-Bildung erfolgt gemaess der Relation $<$.

b) Falls a∈A[der](X[i]), $1 \leq i \leq n$, so sind alle Kanten k[j] durch Regeln q:X[i]→v induziert und es ist ein Besuch des Knotens X[i] erforderlich, um die zu q angegebene Berechnungsvorschrift fuer a auszuwerten. Sei (m,n')=max((up(a[j],down(a[j]+1)), und sei n der kleinste Wert fuer den gilt $n \geq n'$, und fuer kein a'∈A[der](X[l]) gilt (w(a')=(m,n) und i≠l), dann ist w(a)=(m,n). Hierdurch wird sichergestellt, dass die Besuche verschiedener Knoten X[i] und X[l] zu verschiedenen down-Werten der dabei berechneten Attribute fuehren, und dass bei einem Besuch eines Knotens X[i] moeglicherweise mehrere Attribute aus A[der](X[i]) berechnet werden koennen. (Um die Zahl der Besuche einzelner Knoten X[i] klein zu halten, ist es zweckmaessig, ein Attribut nach dieser Regel erst dann zu bewerten, wenn moeglichst viele Attribute a∈A[im](X[i]) bewertet sind.)

c) Falls gilt a∈A[in](Y), so ist zur Berechnung von a ein Besuch des Vorgaenger-Knotens erforderlich. Es ist dann w(a)=(m,∅), wobei m der kleinste Wert ist, fuer den gilt:
$w(a[j]) \leq (m,\emptyset)$, falls a[j]∈A[in](Y), und
$w(a[j]) < (m,\emptyset)$ andernfalls.

9. Aufstellung der Besuchs-Sequenzen

Sind in einem Graphen E[p] zu einer Regel p:Y→X[1]...X[n] alle Attribute bewertet, so kann festgestellt werden, in welcher Reihenfolge die Knoten in der Umgebung eines Knotens des Typs p besucht werden muessen, und welche Attribute zwischen je zwei solchen Besuchen berechnet werden koennen.

Wir stellen zu E[p] eine Besuchs-Sequenz F[p] auf, die eine lineare Folge von Attributen a∈A[der](Y) v A[in](X[i]), $1 \leq i \leq n$ (nur fuer diese gibt es Berechnungs-Vorschriften zu p) und Besuchen b[i,l] ist. Dabei bezeichnet b[i,l] den l-ten Besuch des Knoten X[i], falls $1 \leq i \leq n$, bzw. des Vorgaenger-Knotens von Y, falls i=∅.

F[p] hat die folgenden Eigenschaften:

a) Alle Attribute a∈A[der](Y) v A[in](X[i]), $1 \leq i \leq n$ treten genau einmal in F[p] auf.

b) Fuer je zwei Attribute a[1], a[2] gilt: a[1] steht in F[p] vor a[2], falls $w(a[1]) < w(a[2])$ (Die Anordnung von Attributen mit w(a[1])=w(a[2]) ist bedeutungslos.)

c) Zwischen je zwei Attributen a[1] und a[2] mit $w(a[1]) < w(a[2])$ steht ein Besuch b[i,l]. Es gilt i=∅, falls es ein Attribut a[k] in F[p] vor b[i,l] gibt mit a[k]∈A[in](Y) und w(a[k])=w(a[1]). Anderenfalls gibt es in F[p] vor b[i,l] mindestens ein a[k]∈A[der](X[i]) mit w(a[k])=w(a[1]). Mit l werden alle Besuche des gleichen Knotens (bei 1 beginnend) in der Reihenfolge ihres Auftretens in F[p] durchnumeriert.

d) Gilt fuer das erste Attribut a in F[p] w(a)=(1,Ø), so geht ihr ein Besuch b[Ø,1] voraus.

e) Auf jedes Attribut a∈A[in](X[i]) muss in F[p] ein Besuch b[i,1] folgen.

Die F[p] stellen Vorschriften dar zur Auswertung der Attribute vor Knoten des Typs p.

10. Anpassung der Bewertung von Graphen

Sind alle Graphen E[p] bewertet, so muss ueberprueft werden, ob die Annahme aus Abschnitt 8 gilt, dass alle Attribute a∈A[in](Y), in die in E[p] keine Kanten muenden, beim ersten Besuch des Vorgaenger-Knotens berechnet werden koennen. Trifft dies fuer ein oder mehrere Nichtterminale nicht zu, so sind die Bewertungen der Graphen in geeigneter Weise anzupassen, oder die Grammatik ist zu transformieren.

Sei M die Menge der Attribute a∈A[in](Y), in die in den Graphen E[p] zu Regeln p:Y->u keine Kanten muenden. Zu jeder Regel q:Z->vYw wird nun eine Menge N[q] gebildet mit
N[q] = [a | a∈M und a steht in F[p] nach dem ersten Besuch von Y]
Es sind dann folgende Faelle zu unterscheiden:

a) N[q]=Ø fuer alle q.
Dann gilt die obige Annahme und E[p] und F[p] bleiben unveraendert.

b) Alle N[q] sind gleich und nicht leer.
In diesem Fall muessen die Graphen neu bewertet werden. Dabei erhalten die Attribute a∈M\N das Wertepaar (1,Ø) und a∈N das Wertepaar (2,Ø). Die Bewertung der uebrigen Attribute und die Bestimmung der Besuchs-Sequenz erfolgt wie in den Abschnitten 8 und 9 beschrieben.

c) Es gibt mehrere paarweise verschiedene N[q1],...,N[qm]. In diesem Fall ist die folgende Transformation der Grammatik erforderlich:
Zu jedem N[qj]≠Ø wird die Regel qj:Z[j]->vYw ersetzt durch qj´:Z[j]->vY[j]w, wobei Y[j] ein neues Nichtterminal der Grammatik ist. Zu jeder Regel p:Y->u wird eine neue Regel pj:Y[j]->u eingefuehrt.
Die Graphen E[qj] und die Besuchs-Sequenzen F[qj] sind gleich E[q] bzw. F[q]. Die E[pj] und F[pj] werden wie im Fall b neu bestimmt.

11. Modifikationen des Algorithmus

In Abschnitt 4 wurden Annahmen zur Strategie gemacht, die beim Aufbau und Durchlaufen des AAB anzuwenden ist. Wie zeigen hier, wie der Algorithmus modifiziert werden muss, falls stattdessen die folgenden Annahmen gelten:.

a) Der AAB wird von der Wurzel zu den Endknoten hin aufgebaut. Dabei werden die Knoten zu einer Regel p:Y->X[1]...X[n] in der Reihenfolge Y,T(X[1]),...,T(X[n]) generiert; T(X[i]) bezeichnet den Teilbaum, dessen Wurzel X[i] ist. Damit ist die Reihenfolge der ersten Besuche jedes Knoten festgelegt. Sollen beim ersten Besuch eines Knotens der Klasse p alle Berechnungsvorschriften g[p] ausgewertet werden, die keinen erneuten Besuch des Vorgaenger-Knotens erfordern, so ist nur der Abschntt 8 so zu veraendern, dass die Anfangsbewertung der Attribute a∈A[der](X[i]), in die keine Kanten muenden nicht (∅,∅) sondern (∅,i) ist.

b) Die Auswertung der Berechnungsvorschriften beginnt erst, wenn der AB vollstaendig aufgebaut ist. Der Durchlauf durch den AB beginnt bei seiner Wurzel.
In diesem Fall kann auch die Reihenfolge der ersten Besuche der Knoten X[i] (zu einem Knoten-Typ p:Y->X[1]...X[n]) aus den semantischen Abhaengigkeiten bestimmt werden. Dies wird durch folgende Aenderungen der Regeln zur Bewertung der Graphen (Abschnitt 8) beruecksichtigt:

Die Attribute a∈A[der](X[i]), in die keine Kanten muenden, werden zunaechst nicht bewertet. Stattdessen wird ein Fall d) wie folgt ergaenzt:.

d) Ist schon ein Attribut a[1]∈A[der](X[i]) bewertet, so wird allen a∈A[der](X[i]), in die keine Kanten muenden, das Wertepaar w(a[1]) zugeordnet; andernfalls das Wertepaar (u,d), wobei gilt (u,d-1)=max(w[p](a[1])) und a[1] sind alle schon bewerteten Attribute in E[p]. (Um die Anzahl der Besuche von X[i] gering zu halten, sollte diese Regel moeglichst spaet angewandt werden.)

12. Beispiel

In diesem Abschnitt wird die Anwendung des Verfahrens anhand einer einfachen Programmiersprache demonstriert. Die statische Semantik wurde hier auf die Identifikation vereinbarter Bezeichner und auf eine einfache Bestimmung und Anpassung von Arten beschraenkt. Die Attribute haben folgende Bedeutung:

ACCESS Menge der jeweils gueltigen Bezeichner
DECL Vereinbartes Objekt
PRIMODE. Art vor der Artanpassung
POSTMODE Art nach der Artanpassung

Zu jeder syntaktischen Regel der Grammatik sind die Berechnungsvorschriften fuer die semantischen Attribute, die C-, D- und E-Graphen und die ermittelte Besuchs-Sequenz angegeben. Es wird die folgende Notation verwendet:

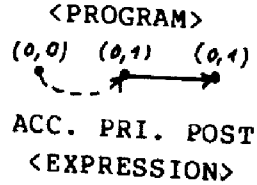
- Nichtterminale werden in <,> eingeschlossen, z.B. <EXPRESSION> .
- Attribute werden durch Nichtterminal.Attribut angegeben (bzw. Nichtterminal.Attribut-ganze Zahl, falls das Nichtterminal mehrfach in der Regel auftritt). z.B. <EXPRESSION>.ACCESS bzw. <EXPRESSION>-1.ACCESS

Die C-,D- und E-Graphen sind jeweils fuer eine Regel zu einem Graphen zusammen gefasst und werden durch die Kennzeichnung der Kanten unterschieden:

---> Kanten im C-,D- und E-Graph
- -> Kanten im D- und E-Graph
...> Kanten im E-Graph

Zu jedem Knoten ist die Bewertung (up,down) des Attributes angegeben.

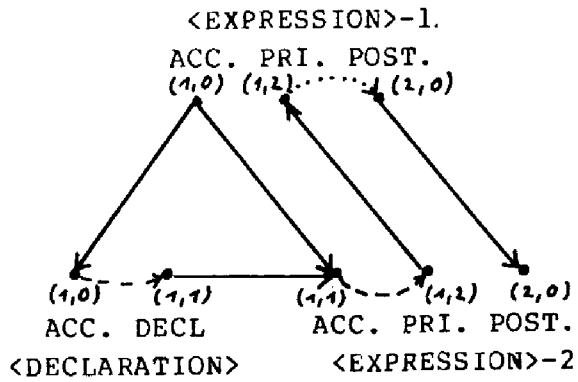
Regel 1:
<PROGRAM> : <EXPRESSION>,
<EXPRESSION>.ACCESS : EMPTY,
<EXPRESSION>.POSTMODE : <EXPRESSION>.PRIMODE.



F[1]=<EXPRESSION>.ACCESS,B[1,1],<EXPRESSION>.POSTMODE,B[1.2]

Regel 2:.

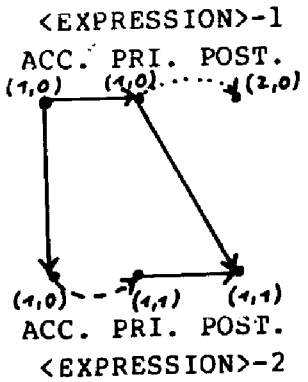
<EXPRESSION> :. BEGIN <DECLARATION> ; <EXPRESSION> END,
<DECLARATION>.ACCESS : <EXPRESSION>-1.ACCESS,
<EXPRESSION>-2.ACCESS :
INCLUDE(<DECLARATION>.DECL,<EXPRESSION>-1.ACCESS),
<EXPRESSION>-1.PRIMODE : <EXPRESSION>-2.PRIMODE,
<EXPRESSION>-2.POSTMODE : <EXPRESSION>-1.POSTMODE.



F[2]=B[0,1],<DECLARATION>.ACCESS,B[1,1],<EXPRESSION>-2.ACCESS,
B[2,1],<EXPRESSION>-1.PRIMODE,B[0,2],<EXPRESSION>-2.POSTMODE

Regel 3:

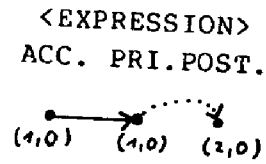
<EXPRESSION> :. IDENTIFIER := <EXPRESSION>,
<EXPRESSION>-2.ACCESS : <EXPRESSION>-1.ACCESS,
<EXPRESSION>-2.POSTMODE :
COERCE(<EXPRESSION>-2.PRIMODE,<EXPRESSION>-1.PRIMODE),
<EXPRESSION>-1.PRIMODE :
IDENTIFY(IDENTIFIER,<EXPRESSION>-1.ACCESS).



F[3]=B[0,1],<EXPRESSION>-2.ACCESS,<EXPRESSION>-1.PRIMODE,B[1,1],
<EXPRESSION>-2.POSTMODE,B[0,2],B[1,2]

Regel 4:

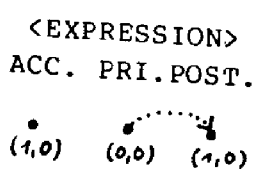
<EXPRESSION> : IDENTIFIER,
    <EXPRESSION>.PRIMODE : IDENTIFY(IDENTIFIER,<EXPRESSION>.ACCESS)



F[4]=B[0,1],<EXPRESSION>.PRIMODE,B[0,2]

Regel 5:

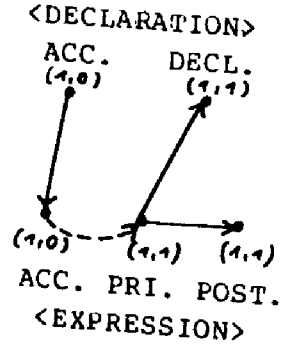
<EXPRESSION> : CONSTANT,
    <EXPRESSION>.PRIMODE : MODE(CONSTANT)



F[5]=<EXPRESSION>.PRIMODE,B[0,1],B[0,2]

Regel 6:.

<DECLARATION> : NEW IDENTIFIER = <EXPRESSION>,
    <DECLARATION>.DECL : DECLARE(IDENTIFIER,<EXPRESSION>.PRIMODE),
    <EXPRESSION>.POSTMODE : <EXPRESSION>.PRIMODE,
    <EXPRESSION>.ACCESS : <DECLARATION>.ACCESS



F[6]=B[0,1],<EXPRESSION>.ACCESS,B[1,1],<DECLARATION>.DECL,
    <EXPRESSION>.POSTMODE,B[1,2]