Zur attributierten Grammatik von PEARL

Von Prof. Dr. S. Heilbrunner und Dipl.-Inform. L. Schmitz, München

1. Einleitung

Der Normenentwurf zur Programmiersprache PEARL [1] enthält eine umfangreiche attributierte Grammatik. Darin eingebettet sind Petri-Netze zur Definition asynchroner Abläufe sowie die Beschreibung der Bedeutung von PEARL-Sprachkonstrukten in "technischem Englisch". Der vorliegende Artikel gibt eine Einführung in attributierte Grammatiken und in die besondere Weise ihrer Verwendung in der PEARL-Norm.

Bei der Definition von Programmiersprachen wird oft eine in Backus-Naur-Form (BNF) niedergeschriebene kontextfreie Grammatik benutzt. Einschränkungen, die sich in BNF nicht formulieren lassen - sogenannte Kontextbedingungen - , werden in natürlicher Sprache angegeben. Attributierte Grammatiken sind Erweiterungen von kontextfreien Grammatiken, die unter anderem die Formulierung von Kontextbedingungen im Rahmen der Grammatik zulassen.

Als Anwendungsbeispiel für diese Einführung dient ein ganz kleiner Ausschnitt aus PEARL, der in Abschnitt 2 mit Hilfe einer kontextfreien Grammatik und zusätzlichen Kontextbedingungen beschrieben wird. In Abschnitt 3 folgt die schrittweise Attributierung dieser kontextfreien Grammatik, wobei informelle Schreibweisen benutzt werden. Der vierte Abschnitt bringt die gleiche Grammatik nochmals, benutzt aber die Schreibweisen des PEARL-Normentwurfs. [1)]

Attributierte Grammatiken wurden eingeführt von D. Knuth [2] und haben seither einen festen Platz unter den Methoden zur formalen Definition von Programmiersprachen. Marcotty, Ledgard und Bochmann [3] geben dazu eine vergleichende Übersicht. Ein weiteres Anwendungsgebiet für attributierte Grammatiken sind übersetzererzeugende Systeme. Eine kurze Einführung in diese Anwendungen gibt Wilhelm [4]. In [3] und [4] findet man auch weitere Literaturhinweise.

2. Ein PEARL-Ausschnitt

Die Produktionsregeln R1-R34 in T a f e l 1 beschreiben einen PEARL-Ausschnitt PA, dessen Programme aus einem Vereinbarungsteil und einem darauf folgenden Zuweisungsteil bestehen. Vereinbart werden einfache Variablen der Typen *fixed* und *float*. Bei Zuweisungen sind beide Seiten einfache Variablen. Die folgende Zeichenreihe ist ein korrektes PA-Programm.

Tafel 1. Die Syntax von PA. Terminale Symbole sind kursiv geschrieben. Die Abkürzungen bedeuten decl(aration), seq(uence), ass(ignment) und var(iable).

```
R1 :  pa-program ::= begin decl-seq ; ass-seq end
R2 :  decl-seq   ::= decl ; decl-seq
R3 :  decl-seq   ::= decl
R4 :  ass-seq    ::= ass ; ass-seq
R5 :  ass-seq    ::= ass
R6 :  decl       ::= dcl var fixed
R7 :  decl       ::= dcl var float
R8 :  ass        ::= var := var
R9 :  var        ::= a
R10:  var        ::= b
. . . . . .
R34:  var        ::= z
```

[1)] Um Mißverständnissen vorzubeugen, sei darauf hingewiesen, daß unsere Grammatik keine Teilgrammatik des Normentwurfs ist.

```
begin
  dcl a fixed; dcl b float; dcl c fixed;
  b := a; c := a; b := c
end
```

Dieses PA-Programm wird uns als laufendes Beispiel durch den Rest des Artikels begleiten.

PA-Programme unterliegen folgenden *Kontextbedingungen:*

KB1: Keine Variable wird mehrmals vereinbart

KB2: Jede benutzte Variable wird vereinbart

KB3: Ist die rechte Seite einer Zuweisung vom Typ *float*, dann auch die linke Seite.

Zur Erläuterung der Kontextbedingungen betrachten wir folgende Zeichenreihe, die den Regeln R1-R34 genügt.

```
begin
  dcl a fixed; dcl a fixed; dcl c float;
  b := a; a := c;
end
```

Die Variable *a* wird zweimal vereinbart im Widerspruch zu KB1. Die Zuweisung *b := a* widerspricht KB2, die Zuweisung *a := c* widerspricht KB3.

Neben Kontextbedingungen gibt es noch implementierungsabhängige Einschränkungen von Übersetzern und Rechensystemen oder auch Einschränkungen der Sprache (PEARL) auf eine Teilmenge (BASIC-PEARL). Wir versehen deshalb PA mit der *Implementierungsbedingung*

IB: Es dürfen höchstens anzvar Variablen vereinbart werden, wobei anzvar eine implementierungsabhängige Konstante ist.

Außerdem benutzen wir die (etwas künstliche) *Teilmengenbedingung*

TB: In BASIC-PA hat die linke Seite jeder Zuweisung den gleichen Typ wie die rechte Seite.

Unser laufendes Beispiel verletzt TB und erfüllt IB, falls anzvar ≥ 3 gilt.

3. Attributierte Grammatiken

Wir wollen zeigen, wie sich die Bedingungen aus Abschnitt 2 in die Grammatik für PA einbringen lassen, das heißt, mit den Regeln R1-R34 verknüpfen lassen. Zunächst ordnen wir jeder Zeichenfolge, die sich mit der kontextfreien Grammatik herstellen läßt, ihren *Strukturbaum* zu. B i l d 1 zeigt den Baum für das laufende Beispiel.

In den nächsten Abschnitten versehen wir die Knoten des so gewonnenen Strukturbaumes mit gewissen Werten - sogenannten *Attributwerten*, kurz: *Attribute* - und prüfen die Bedingungen KB1-KB3, IB und TB anhand dieser Werte nach.

3.1. Attributierung des Beispiels

Um die Kontextbedingungen KB2 und KB3 nachprüfen zu können, muß man bei jeder Zuweisung wissen, welche Variablen vereinbart wurden und welche Typen sie haben. Wir versehen deshalb alle ass-Knoten des Strukturbaumes mit der Liste der vereinbarten Variablen und deren Typen. Im Beispiel ergibt das die Liste

(<*a*,*fixed*>,<*b*,*float*>,<*c*,*fixed*>)

für die Knoten 10,22 und 35. Weiter versehen wir die var-Knoten mit den daraus abgeleiteten Variablenbezeichnern als Attributwerten. Für die Knoten 19 und 21 erhalten wir *b*, bzw. *a*.

Für den bei ass_{10} beginnenden Teilbaum ergibt die Attributierung B i l d 2.

Nach dieser Attributierung können wir eine neue Fassung von KB2 angeben. Sie wird an die Produktionsregel

R8: ass ::= var := var

angefügt und lautet:

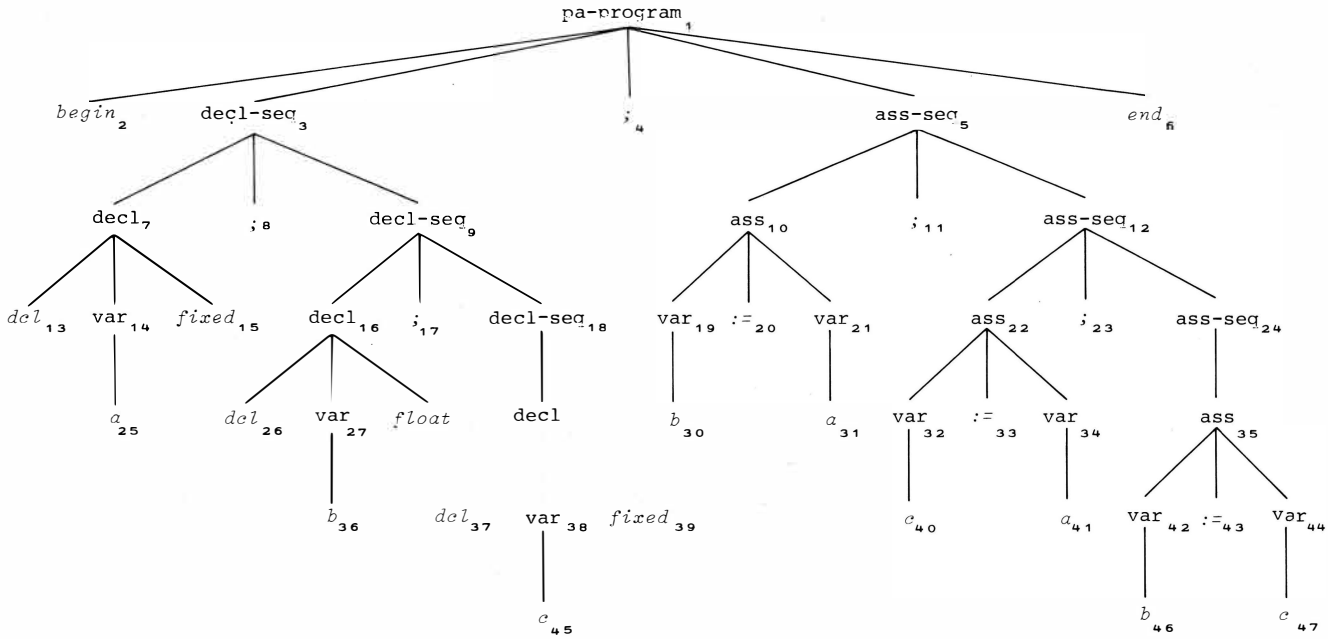

Bild 1. Strukturbaum zum laufenden Beispiel. Die Indizes numerieren die Knoten zur besseren Unterscheidung

KB2': Die Attributwerte des linken und des rechten var-Knotens kommen im Attributwert des ass-Knotens vor.

Man prüft KB2' für B i l d 2 sofort nach. KB2' muß im attributierten Strukturbaum überall dort gelten, wo R8 verwendet wurde. Die Formulierung von KB2' ist noch etwas umständlich. Das liegt vor allem daran, daß wir die Attribute nicht benennen können. Wir führen deshalb *Attributnamen* ein. Die Attributwerte der ass-Knoten nennen wir env-Attributwerte (nach engl.environment: Umgebung) und schreiben ass.env für den env-Attributwert der ass-Knoten. Man sieht, Attribute können als Komponenten der Knoten des Strukturbaumes verstanden werden.

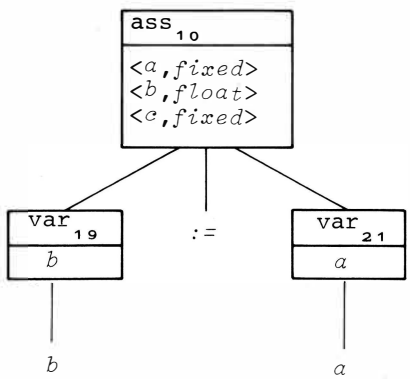

Bild 2. Attributierter Teilbaum für ass_{10}.

Entsprechend verfahren wir mit den var-Knoten. Ihre Attribute bezeichnen wir mit var.denot (für engl. denotation: Bezeichnung). In der Produktionsregel R8 kommt var zweimal vor. Um Eindeutigkeit zu erreichen, numerieren wir gleiche syntaktische Variablen in den Produktionsregeln von links nach rechts und erhalten damit

var[1].denot bzw. var[2].denot

als Bezeichnung für die denot-Attribute des ersten bzw. zweiten Auftretens von var in R8.

Für KB2 ergibt sich damit in Zusammenhang mit der Regel

R8: ass ::= var := var

schließlich die Formulierung

KB2": ass.env enthält Paare <var[1].denot,...> und <var[2].denot,...>.

Wir benutzten jetzt auch in KB3 Attributwerte und fügen an R8 noch an:

KB3': Enthält ass.env das Paar <var[2].denot,*float*> , dann auch das Paar <var[1].denot,*float*>

Man prüft leicht nach, daß an allen drei Stellen, wo R8 im Strukturbaum des laufen-

den Beispiels benutzt wird, die Bedingungen KB2" und KB3' erfüllt sind. Man betrachte hierzu die entsprechenden Ausschnitte aus B i l d 1 . Schließlich fassen wir noch die Teilmengenbedingung neu und fügen sie an R8 an als

TB': Kommt <var[2].denot,*fixed*> in ass.env vor, dann auch <var[1].denot,*fixed*> (weiter ist wegen KB3' nichts zu fordern).

Bis jetzt wurde gezeigt, wie Attributwerte bei kontextfreien Regeln benannt werden, und wie sie zum Abfassen von Bedingungen benutzt werden können. Wie aber berechnet man Attributwerte? Dazu werden den Produktionsregeln neben den Bedingungen noch *Attributauswertungsregeln* beigefügt, die an jeder Stelle des Strukturbaums anzuwenden sind, an der die Produktionsregel benutzt wird. Attributauswertungsregeln haben die Form von Zuweisungen an Attributnamen.

Im Beispiel wird der Produktionsregel

R9: var ::= *a*

die Attributauswertungsregel

var.denot := *a*

mitgegeben. Die Regeln R10-R34 erhalten analog dazu die Attributauswertungsregeln

var.denot := *b* ..... var.denot := *z*

Damit können wir alle var-Knoten attributieren.

Schwieriger ist die Bestimmung der env-Attribute. Wir nehmen an, der Knoten $ass\text{-}seq_5$ habe bereits sein env-Attribut. Dann lassen sich die Attributwerte der Söhne von $ass\text{-}seq_5$ berechnen, indem wir an die Regel

R4: ass-seq ::= ass ; ass-seq

die Attributauswertungsregeln

ass.env := ass-seq[1].env ,
ass-seq[2].env := ass-seq[1].env

anfügen und an

R5: ass-seq ::= ass

die Attributauswertungsregel

ass.env := ass-seq.env

Mit den bisherigen Attributauswertungsregeln können wir den Zuweisungsteil von PA-Programmen vollständig attributieren. Wir erhalten den Teilbaum aus B i l d 3 .

Wir attributieren jetzt den Vereinbarungsteil und betrachten dazu B i l d 4 , das einen Ausschnitt aus dem Strukturbaum zeigt. Um mehrfache Vereinbarungen von Variablen sofort erkennen zu können, müssen wir für jeden decl-Knoten die Liste der links davon vereinbarten Variablen kennen. Zur Aufbewahrung dieser Liste geben wir den decl-Knoten das Attribut pre-env. Die zu einem decl-Knoten gehörige Vereinbarung verlängert dieses Attribut um einen Eintrag und erzeugt das Attribut des nächsten decl-Knotens. Zur Aufbewahrung der verlängerten Listen geben wir jedem decl-Knoten noch ein zweites Attribut, das post-env-Attribut. Für die decl-Knoten aus B i l d 4 erhalten wir damit folgende Attributierungen, wobei () die leere Liste bezeichnet.

$decl_7$	
pre-env:	post-*env*:
()	<*a*,*fixed*>

$decl_{16}$	
pre-env:	post-env:
<*a*,*fixed*>	<*a*,*fixed*> <*b*,*float*>

$decl_{29}$	
pre-env:	post-env:
<*a*,*fixed*> <*b*,*float*>	<*a*,*fixed*> <*b*,*float*> <*c*,*fixed*>

Wie lauten die zugehörigen Attributauswertungsregeln? Im Beispiel (vgl. B i l d 4) können Attribute von $decl_{16}$ nicht unmittelbar an $decl_{29}$ übergeben werden, da es keine Produktionsregel gibt, deren Anwendung $decl_{16}$ und $decl_{29}$ gleichzeitig betrifft.

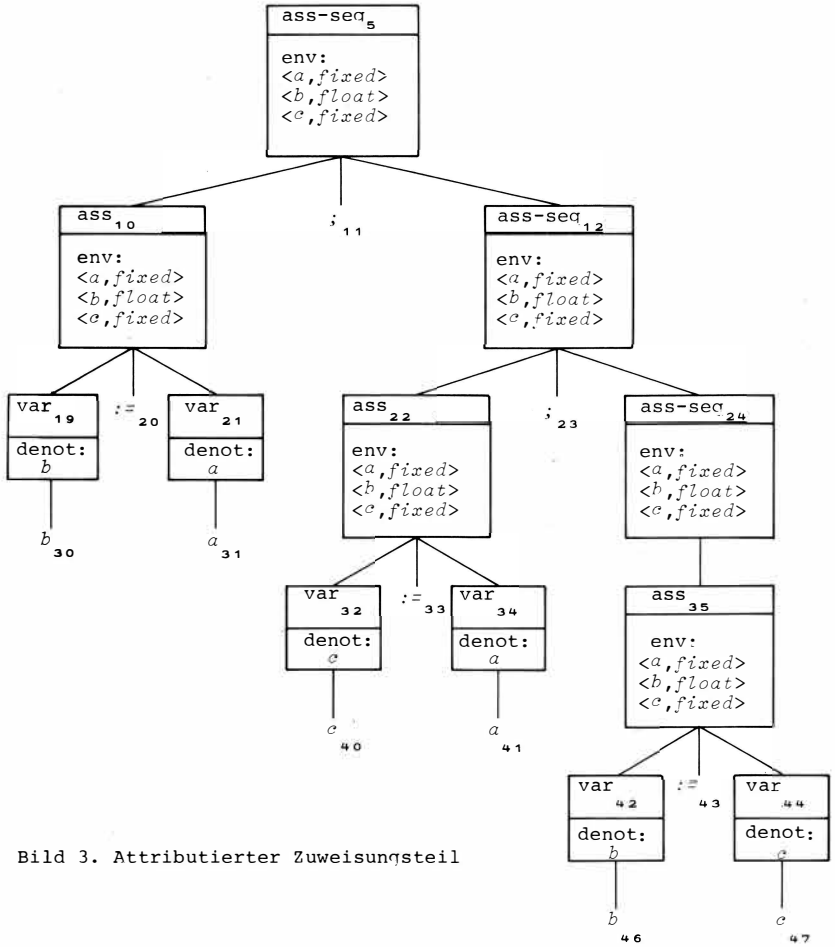

Bild 3. Attributierter Zuweisungsteil

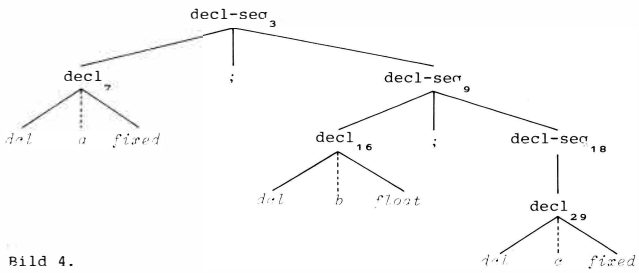

Bild 4.

Vielmehr müssen Attribute von $decl_{16}$ über die Knoten $decl\text{-}seq_9$ und $decl\text{-}seq_{18}$ zu $decl_{29}$ transportiert werden. Zur Abwicklung dieses Transports versehen wir auch die decl-seq-Knoten mit den Attributen pre-env und post-env. Das ergibt B i l d 5.

Aus diesem Bild lassen sich die Attributauswertungsregeln sofort ablesen. Für

R2: decl-seq ::= decl ; decl-seq

ergeben sich

decl.pre-env := decl-seq[1].pre-env ,
decl-seq[2].pre-env := decl.post-env ,
decl-seq[1].post-env := decl-seq[2].post-env.

Für

R3: decl-seq ::= decl

ergeben sich

decl.pre-env := decl-seq.pre-env ,
decl-seq.post-env := decl.post-env .

Wir erinnern daran, daß die var-Knoten bereits das var.denot-Attribut besitzen und versehen

R6: decl ::= *dcl* var *fixed*

mit

decl.post-env := decl.pre-env
+<var.denot,*fixed*>

und

R7: decl ::= *dcl* var *float*

mit

decl.post-env := decl.pre-env
+<var.denot,*float*>

In den letzten zwei Attributauswertungs-

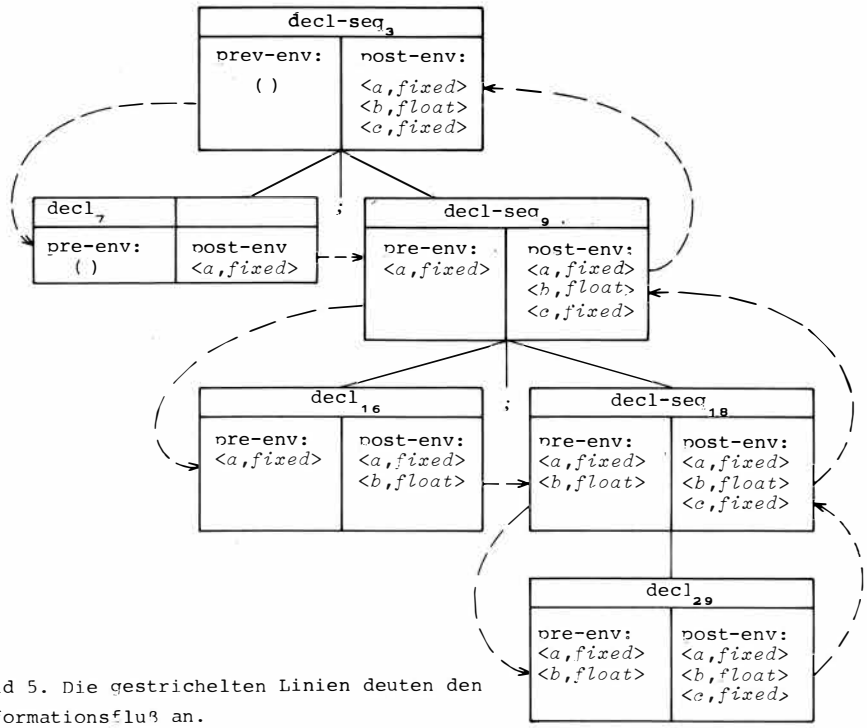

Bild 5. Die gestrichelten Linien deuten den Informationsfluß an.

regeln ist "+" der Konkatenationsoperator für Listen.

Es fehlen noch zwei Attributierungsregeln. Die erste führt die ursprünglich leere Umgebung ein. Die zweite reicht die im Vereinbarungsteil aufgesammelte Liste von Variablen an den Zuweisungsteil weiter. Das ergibt für

R1: pa-programm ::= *begin* decl-seq ;
ass-seq *end*

die Attributierungsregeln

decl-seq.pre-env := ()
ass-seq.env := decl-seq.post-env

Insgesamt erhalten wir für den Vereinbarungsteil den attributierten Baum aus B i l d 6 .

An R1 schließt sich noch die Implementierungsbedingung IB an als

IB': Die Liste decl-seq.post-env enthält höchstens anzvar Einträge.

3.2. Begriffliches

Bei der Attributierung des Beispiels im letzten Abschnitt sind eine Reihe von Fragen offen geblieben, die wir nun diskutieren wollen.

Was bedeutet es, wenn eine Bedingung nicht erfüllt ist? Wenn bei der Attributierung des Strukturbaumes zu einer vorgelegten Zeichenreihe Z die Kontextbedingungen überall dort erfüllt sind, wo die zugehörigen kontextfreien Regeln zum Aufbau des Strukturbaumes verwendet wurden, dann und nur dann ist Z ein korrektes PA-Programm. Sind außerdem alle Teilmengenbedingungen erfüllt, dann ist Z ein korrektes BASIC-PA-Programm. Verletzt ein korrektes (BASIC-) PA-Programm eine Implementierungsbedingung, dann genügt Z nicht den Einschränkungen der betreffenden Implementierung.

In welcher Reihenfolge werden Attribute ausgewertet? Grundsätzlich wird jeder Attributwert genau einmal bestimmt. Die Reihenfolge der Attributauswertungen ist beliebig, unterliegt aber der Einschränkung,

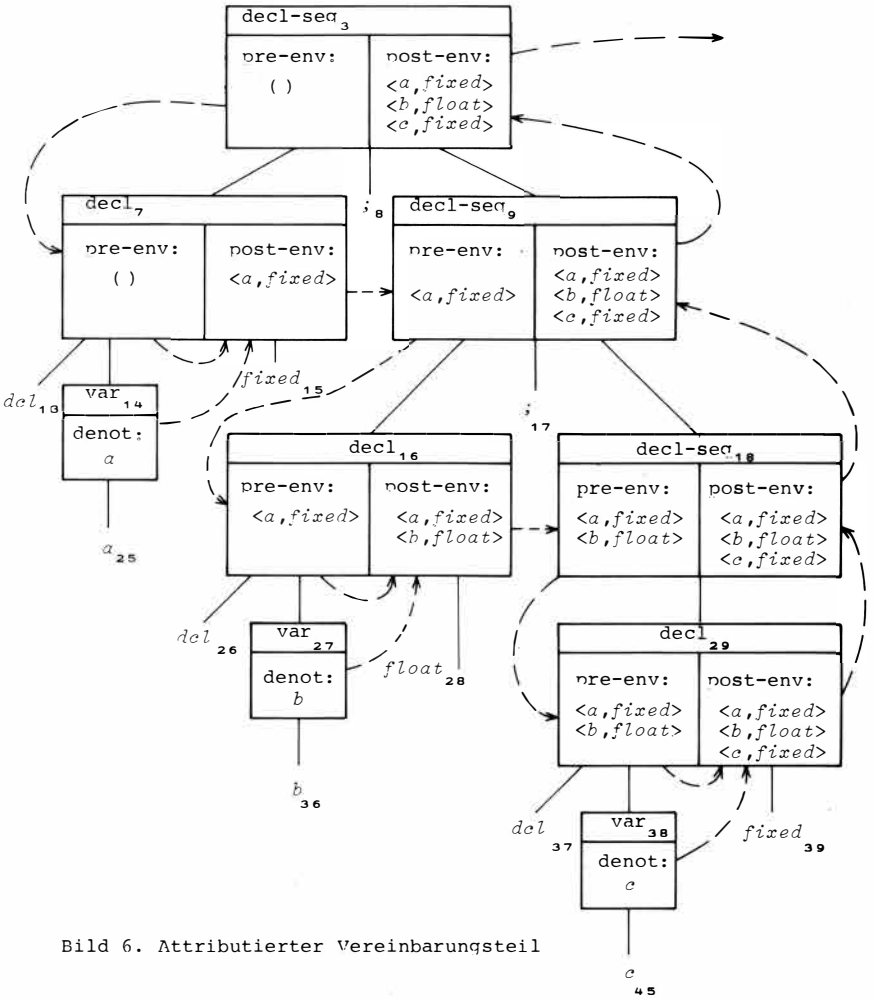

Bild 6. Attributierter Vereinbarungsteil

daß eine Attributauswertungsregel erst dann anwendbar ist, wenn die auf der rechten Seite vorkommenden Attributwerte bereits berechnet sind. Für PA-Programme können die Attribute in einem Durchlauf des Strukturbaumes von links nach rechts ausgewertet werden. Bei praktischen Programmiersprachen ist das in der Regel nicht der Fall, und zwar aus dem gleichen Grund, aus dem diese Programmiersprachen Mehr-Pass-Übersetzer erfordern.
Im allgemeinen gilt deshalb für die Attributierung folgende Vorschrift:

V: Solange noch nicht alle Attributwerte eines Strukturbaumes bestimmt sind, suche eine anwendbare Attributauswertungsregel und wende sie an.

Diese Vorschrift führt nur dann zu einer vollständigen und eindeutigen Attributierung von Strukturbäumen, wenn die attributierte Grammatik wohldefiniert [2] ist.

Hätten wir z.B. im letzten Abschnitt eine der benötigten Attributauswertungsregeln vergessen, dann wäre die vollständige Attributierung des Beispiels offenbar nicht möglich gewesen. Hätten wir dagegen zu viele Attributauswertungsregeln angegeben, wären u.U. für den gleichen Strukturbaum verschiedene Attributierungen möglich gewesen. Eine andere Fehlersituation hängt mit den Abhängigkeiten zwischen Attributwerten zusammen, die wir uns wie im B i l d 6 durch gestrichelte Pfeile gegeben denken: Bilden solche gestrichelte Pfeile in einem Strukturbaum einen geschlossenen *Zyklus*, dann läßt sich keines der auf dem Zyklus liegenden Attribute auswerten, da man zur Auswertung eines jeden Attributs den Wert eines anderen, ebenfalls auf dem Zyklus gelegenen Attributs benötigt.

Gibt es zu der Produktionsregel A ::= ...B... eine Attributauswertungsregel A.x := ..., dann heißt das Attribut x von A *synthetisiert berechnet*. Gibt es zu der Produktionsregel eine Attributauswertungsregel B.y := ..., dann heißt das Attribut y von B *ererbt berechnet*.

Mit Hilfe dieser Begriffe können wir Bedingungen dafür angeben, wann eine attributierte Grammatik wohldefiniert [2] ist:

(i) Alle Knoten zur gleichen syntaktischen Variablen haben Attribute gleichen Namens. [1]

(ii) Ist x Name eines Attributs einer syntaktischen Variablen A, so wird A.x stets entweder synthetisiert oder ererbt berechnet und ist damit entweder ein *synthetisiertes* oder ein *ererbtes* Attribut von A (T a f e l 2). Das Startsymbol der Grammatik hat keine ererbten Attribute.

Tafel 2. Attribute von PA

syntaktische Variable	Attribute	
	ererbt	synthetisiert
decl-seq	pre-env	post-env
decl	pre-env	post-env
ass-seq	env	
ass	env	
var		denot

(iii) In jeder Produktionsregel gibt es zu jedem synthetisierten Attribut der linken Seite und zu jedem ererbten Attribut der rechten Seite genau eine Attributauswertungsregel, d.h. die Menge der Attributauswertungsregeln ist vollständig.

(iv) In keinem Strukturbaum hängen Attributwerte zyklisch voneinander ab.

Im allgemeinen ist Zyklenfreiheit nur mit erheblichem Aufwand feststellbar. In besonderen Fällen, wie zum Beispiel bei PEARL, läßt sie sich jedoch mit vertretbarem Aufwand beweisen.

4. ALADIN

Will man eine umfangreiche attributierte Grammatik übersichtlich und vollständig niederschreiben, so erweist sich dabei ein strenges Schema als zweckmäßig. Die in der PEARL-Beschreibung [1] verwendete Sprache ALADIN bietet ein solches Schema an.

Die attributierte PEARL-Grammatik gliedert sich in mehrere, ähnlich aufgebaute Teilgrammatiken. Im Anhang bringen wir die attributierte Grammatik zu PA im Stile einer solchen Teilgrammatik (vgl. z.B.[1], S. 125-162). Wir erläutern nun die darin vorkommenden ALADIN-Sprachelemente.

Im letzten Abschnitt haben wir *Attributtypen* -- das sind die Wertebereiche, zu denen Attributwerte gehören -- informell eingeführt, z.B. als "Listen der vereinbarten Variablen und deren Typen". ALADIN verlangt die Vereinbarung von Attributtypen in Anlehnung an die Vereinbarung von Datentypen in höheren Programmiersprachen. Die ALADIN-Beschreibung des Typs tp_env von env lautet

```
TYPE tp_env : LISTOF tp_pair
```

Der Attributtyp tp_pair ist dabei zu definieren durch

```
TYPE tp_pair:  STRUCT(s_denot: tp_denot,
                      s_kind: tp_kind)
```

Das heißt, Werte des Attributtyps tp_pair sind zusammengesetzt ("STRUCT") aus Werten vom Typ tp_denot und tp_kind. Als Selektoren für die Komponenten werden dabei s_denot und s_kind verwendet. Ist etwa p

[1] Der PEARL-Normentwurf verwendet eine etwas schwächere Bedingung, die allerdings schwerer zu formulieren ist.

vom Typ tp_pair, so ist p.s_denot seine erste Komponente.

Die Attributtypen tp_denot und tp_kind sind skalar und werden durch Aufzählung ihrer Elemente definiert.

```
TYPE tp_denot: (sc_A,sc_B,...,sc_Z) ,
TYPE tp_kind: (sc_FIXED,sc_FLOAT) .
```

Weitere Möglichkeiten zur Bildung von Attributtypen finden sich in [1], S.15 f.

Den Namen sind Präfixe vorangestellt, die als Lesehilfe gedacht sind. "tp_" kennzeichnet Namen von Attributtypen, "s_" wird bei Selektoren verwendet, "at_" bei Attributnamen und "sx_" bei syntaktischen Variablen (vgl. [1], S. 14).

Konstantendefinitionen führen Namen für Werte ein. Das dient vor allem der Abkürzung und der Lesbarkeit. Außerdem treten dadurch implementierungsabhängige Werte nur bei den Konstantendefinitionen auf und können deshalb für jede Implementierung leicht ergänzt werden. Wir benutzen die Konstanten c_imp_anzvar und c_non_BASIC_PA in diesem Sinn.

Jede *syntaktische Variable* muß vereinbart werden. Dabei sind die Namen und Typen ihrer Attribute anzugeben. Ererbte Attribute sind durch den Zusatz "INH" (für engl. inherited) gekennzeichnet. Die synthetisierten Attribute werden nicht besonders gekennzeichnet.

ALADIN-Regeln enthalten neben einer Produktionsregel die zugehörigen Bedingungen und Attributauswertungsregeln. Im folgenden Schema dürfen die nicht benötigten Teile weggelassen und die CONDITION-Teile wiederholt werden (vgl.[1], z.B. S. 44 und 134 f).

```
RULE    Produktionsregel
   SUBSET
      CONDITION  Teilmengenbedingung
   STATIC
      Attributauswertungsregeln
      CONDITION Kontext- oder Implementierungsbedingung
   DYNAMIC
      Beschreibung der Wirkung von
      Sprachelementen in "technical
      English"
END
```

ALADIN erlaubt die Definition von *Funktionen* und ihre Verwendung in Ausdrücken. Ihre Definition lehnt sich an die Vereinbarung von Funktionsprozeduren in höheren Programmiersprachen an (vgl.[1], S. 23). ALADIN stellt eine Reihe vordefinierter Funktionen zur Verfügung (vgl.[1],S.23), die für unseren PEARL-Ausschnitt ausreichen.

Die env-Attribute sind Listen von Paaren, die jeweils aus einem Variablennamen und dem zugehörigen Typ bestehen. Da sich die Paare durch die Variablennamen unterscheiden, erklären wir Variablennamen zu *Schlüsseln* ("KEY"), indem wir "KEY s_denot" bei der Definition des Attributtyps tp_env anfügen. Diese Konvention spielt bei der Verwendung der vordefinierten Funktionen eine Rolle.

Wir benötigen folgende vordefinierte Funktionen:

LENGTH(p_list) = Länge der Liste p_list.

KEY_IN_LIST(p_key,p_list) = "wahr", falls p_list ein Element mit dem Schlüssel p_key enthält, und "falsch" sonst.

SELECT_BY_KEY(p_key,p_list) = das Glied von p_list mit dem Schlüssel p_key, falls es ein solches gibt und es eindeutig bestimmt ist, undefiniert sonst.

Der Anhang enthält die in ALADIN formulierte Grammatik zu PA.

Literatur

[1] DIN 66 253, Teil 2: Programmiersprache PEARL, FULL PEARL. Berlin, 1980.

[2] K n u t h, D.E.: Semantics of context-free languages. Math. System Theory 2 (1968) S. 127-145 *und* Math. Systems Theory 5 (1971) S. 95.

[3] M a r c o t t y, M., L e d g a r d, H.F., B o c h m a n n, G.V.: A Sampler of Formal Definitions. Computing Surveys Vol. 8, No. 2 (1976) S. 191-276.

[4] W i l h e l m, R.: Attributierte Grammatiken. Informatik-Spektrum Bd. 2, No. 3 (1979) S. 123-130.

ANHANG

Attribute Types

```
TYPE tp_env:    LISTOF tp_pair KEY s_denot;
TYPE tp_pair:   STRUCT(s_denot: tp_denot, s_kind: tp_kind);
TYPE tp_denot:  (sc_A,sc_B,...,sc_Y,sc_Z);
TYPE tp_kind:   (sc_FIXED,sc_FLOAT);
```

Constants

```
CONST c_imp_anzvar: 10;
    % Kommentarzeilen, wie diese beginnen mit dem Prozentzeichen. Anstelle von 10
    % hätte man auch jede andere Zahl wählen können, z.B. 255.
CONST c_non_BASIC_PA: TRUE;
    % Mit c_non_BASIC_PA = TRUE  ist die Teilmengenbedingung immer wahr. Ersetzt man
    % TRUE durch FALSE, so ist die Teilmengenbedingung nur dann wahr, wenn die
    % Zeichenfolge ein BASIC-PA-Programm ist.
```

Symbols

```
NONTERM sx_PA-program: ;
NONTERM sx_decl_seq: at_pre_env: tp_env INH, at_post_env: tp_env;
NONTERM sx_ass_seq:  at_env: tp_env INH;
NONTERM sx_decl:     at_pre_env: tp_env INH, at_post_env: tp_env;
NONTERM sx_ass:      at_env: tp_env INH;
NONTERM sx_var:      at_denot: tp_denot;
```

Rules

```
RULE sx_PA_program ::=  'BEGIN' sx_decl_seq ';' sx_ass_seq 'END'
   % Terminale Zeichen sind in Apostrophe einzuschließen.
   STATIC sx_ass_seq.at_env := sx_decl_seq.at_post_env;
          sx_decl_seq.at_pre_env := tp_env()
           % leere Liste vom Typ tp_env.
   CONDITION LENGTH(sx_ass_seq.at_env)≤·c_imp_anzvar
           % Implementierungsbedingung
END;
```

```
RULE sx_decl_seq ::= sx_decl ';' sx_decl_seq
   STATIC sx_decl_seq[1].at_post_env := sx_decl_seq[2].at_post_env;
          sx_decl.at_pre_env := sx_decl_seq[1].at_pre_env;
          sx_decl_seq[2].at_pre_env := sx_decl.at_post_env;
END;

RULE sx_decl_seq ::= sx_decl
   STATIC sx_decl_seq.at_post_env := sx_decl.at_post_env;
          sx_decl.at_pre_env := sx_decl_seq.pre_env
END;

RULE sx_ass_seq ::= sx_ass ';' sx_ass_seq
   STATIC sx_ass.at_env := sx_ass_seq[1].at_env;
          sx_ass_seq[2].at_env := sx_ass_seq[1].at_env
END;

RULE sx_ass_seq ::= sx_ass
   STATIC sx_ass.at_env := sx_ass_seq.at_env
END;

RULE sx_decl ::= 'DCL' sx_var 'FIXED'
   STATIC sx_decl.at_post-env := sx_decl.at_pre_env +
                                 tp_env(tp_pair(sx_var.at_denot,sc_FIXED))
          % Der Operator + verkettet zwei Listen gleichen Typs -- hier des Typs tp_env.
          % tp_pair(...,...) setzt die beiden Argumente zu einem Paar zusammen und
          % tp_env(...) macht daraus eine einelementige Liste.
   CONDITION NOT KEY_IN_LIST(sx_var.at_denot,sx_decl.at-pre_env)
          % Kontextbedingung KB1
END;

RULE sx_decl ::= 'DCL' sx_var 'FLOAT'
     wie oben aber mit FLOAT statt FIXED
END;

RULE sx_ass ::= sx_var ':=' sx_var
   SUBSET CONDITION c_non_BASIC_PA OR
             SELECT_BY_KEY(sx_var[1].at_denot,sx_ass.at_env).s_kind =
             SELECT_BY_KEY(sx_var[2].at_denot,sx_ass.at_env).s_kind
             % Teilmengenbedingung. Beachte die logische Äquivalenz von
             % A V B  und  ¬A→B
   STATIC CONDITION SELECT_BY_KEY(sx_var[2].at_denot,sx_ass.at_env).s_kind = sc_FIXED
               OR   SELECT_BY-KEY(sx_var[1].at_denot,sx_ass.at_env).s_kind = sc_FLOAT
             % Kontextbedingung KB3
      CONDITION KEY_IN_LIST(sx_var[1].at_denot,sx_ass.at_env) AND
                KEY_IN_LIST(sx_var[2].at_denot,sx_ass.at_env)
                % Kontextbedingung KB2
END;

RULE sx_var ::= 'A'
   STATIC  sx_var.at_denot := sc_A
END;
Weitere Regeln sind zu ergänzen für B,C,...,Y,Z.
```