Nutzen attributierter Grammatiken in Sprachdefinitionen

von Prof. Dr. S. Heilbrunner, München

1. Einleitung

Mit der Norm einer Programmiersprache sind eine Reihe von Erwartungen verbunden. Sie soll widerspruchsfrei sein, vollständig und genau, soll für den interessierten Anwender gut lesbar sein und soll daneben noch dem Implementierer als Ausgangspunkt für einen Übersetzer dienen. Diesen Erwartungen, die mit den heute bekannten Mitteln wohl kaum gleichzeitig befriedigt werden können, stellte sich der für PEARL [1] zuständige Ausschuß und entschied sich für Genauigkeit bei leidlicher Lesbarkeit als oberstes Ziel.

Zur Beschreibung der syntaktischen Struktur von Programmiersprachen sind kontextfreie Grammatiken heute ohne Konkurrenz. Zur Beschreibung semantischer Eigenschaften stehen mehrere Methoden zur Wahl [5], von denen aber noch keine die Feuerprobe der Verwendung im definierenden Dokument einer ernsthaften Programmiersprache bestanden hat. Denn vom Algol 68 Bericht, dem einzigen nennenswerten Versuch in dieser Richtung, kann man ruhigen Gewissens sagen, daß er gegen das darin verwendete Hilfsmittel spricht. Angesichts dieses monumentalen Fehlschlags ist der Mut der Erfinder von PEARL zu bewundern, mit dem sie sich für eine formale Methode entschieden.

Die wünschenswerten Eigenschaften der gesuchten formalen Methode füllen einen langen Zettel. Leider ist es utopisch zu hoffen, daß sich ein Hilfsmittel findet, das auch bei sorglosem Umgang Widerspruchsfreiheit und Vollständigkeit in realistischen Anwendungen garantiert. Es entspricht vielmehr dem Stand der Technik, daß sich bei Beachtung nicht-trivialer Bedingungen mit mehr oder weniger aufwendigen Algorithmen Widerspruchsfreiheit und Vollständigkeit in Teilen nachweisen lassen. So verbleibt als größter Vorteil der formalen Hilfsmittel, daß sie die Darstellung verkürzen und besser gliedern. Dadurch wird wie beim Programmieren bessere Übersicht erzielt, die Fehler vermeiden hilft.

Zwei Methoden standen zur Wahl: Konjugationsgrammatiken und attributierte Grammatiken. In beiden Fällen sollten zusätzlich Petri-Netze benutzt werden. Wenn auch die Hoffnung auf bessere Lesbarkeit eine Rolle spielte, so scheint die Entscheidung für eine attributierte Grammatik doch mehr auf eher äußeren Umständen denn auf nachgewiesener Überlegenheit gegenüber Konjugationsgrammatiken zu beruhen.

2. Attributierte Grammatiken

"Attributierte Grammatik" ist ein schillernder Begriff, der eine Vielfalt von Ausformungen umfaßt. Wegen ihrer überragenden Rolle im Übersetzerbau scheint sich die Vorstellung verbreitet zu haben, attributierte Grammatiken seien immer praxisorientiert. Das ist ein Mißverständnis. Je nach Handhabung lassen sich mit ihnen ganz unterschiedliche Ziele erreichen, was im folgenden an einem Beispiel erläutert werden soll. Das Beispiel kann weder eine Einführung in attributierte Grammatiken ersetzen [2,5] noch eine Einführung in den Übersetzerbau [4].

2.1. Ein Beispiel

Als Beispiel benutzen wir die Sprache, die durch die Produktionsregeln in T a f e l 1 definiert ist. Programme in dieser

Tafel 1. Die Produktionsregeln der Beispielsprache. Für die syntaktische Variable id sind geeignete Produktionsregeln zu ergänzen.

programm	::=	*begin* vereinbarungen; zuweisungen *end*
vereinbarungen	::=	vereinbarungen ; vereinbarung
vereinbarungen	::=	vereinbarung
vereinbarung	::=	*dcl* id *fixed*
vereinbarung	::=	*dcl* id *string*
zuweisungen	::=	zuweisungen ; zuweisung
zuweisungen	::=	zuweisung
zuweisung	::=	id := ausdruck
ausdruck	::=	id + id

Sprache haben einen Vereinbarungs- und Zuweisungsteil. Vereinbart werden Variablen der Typen fixed und string. Die rechten Seiten der Zuweisungen sind einfache Ausdrücke, wobei "+" die Addition von Zahlen, aber auch die Konkatenation von Zeichenreihen bedeutet. Die üblichen Kontextbedingungen werden gefordert: keine doppelten Vereinbarungen, verträgliche Typen bei Zuweisungen, usw.

Das folgende Programm genügt diesen Bedingungen und wird uns als laufendes Beispiel dienen.

```
begin
  dcl a fixed; dcl b string; dcl c string;
  a := a+a; c := b+b
end
```

Wir wollen die Kontextbedingungen mit einer attributierten Grammatik beschreiben, die sich an die Regeln von Tafel 1 anlehnt. Wir benutzen zwei Wege. Beim ersten ist das Ziel die klare Formulierung der Attributberechnung. Beim zweiten soll dem Leser ein Eindruck davon vermittelt werden, welche Annahmen und Hilfsmittel benutzt werden, um die für Übersetzer nötige Effizienz der Attributauswertung zu erreichen. Jeder Attributierung liegen die Strukturbäume zugrunde, die sich aus der Anwendung der Produktionsregeln ergeben (vgl. B i l d 1).

2.2. Attributierung des Beispiels

Um im Anweisungsteil die vereinbarten Variablen zu kennen, sammeln wir sie im Vereinbarungsteil zusammen mit ihren Typen auf. Wir attributieren deshalb die Vereinbarungsknoten mit den Listen der darunter liegenden Variablen und geben ihnen dazu das synthetisierte Attribut umgebung (vgl. B i l d 2). Tafel 2 gibt eine Übersicht über die Attribute. Die Attributauswertungsregeln für den Vereinbarungsteil finden sich in T a f e l 3. Die dort benutzte Funktion ENTHALTEN hat die naheliegende Bedeutung, so gelten zum Beispiel

ENTHALTEN (*a*, {<*a*, *fixed*>, <*b*, *string*>}) = *true*

und

ENTHALTEN (*c*, {<*a*, *fixed*>, <*b*, *string*>}) = *false*

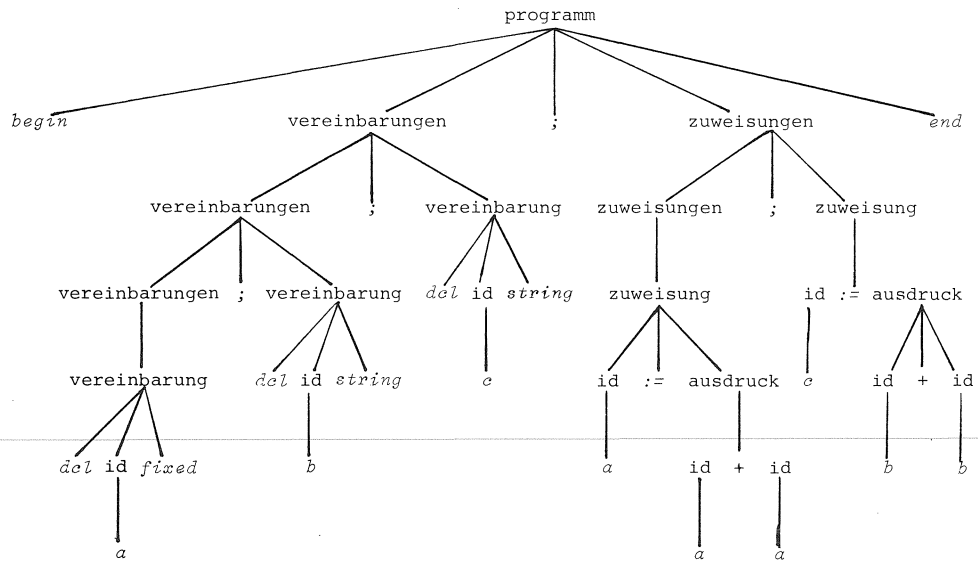

Bild 1. Strukturbaum zum laufenden Beispiel

Tafel 2. Attribute der Syntaxvariablen.
s steht für synthetisiert,
e für ererbt.

syntaktische Variable	Attribute	Art	Bedeutung
vereinbarungen	umgebung	s	Liste der vereinbarten Variablen nebst Typen
vereinbarung	eintrag	s	Paar aus vereinbarter Variablen und Typ
id	bezeichner	s	aus id abgeleiteter Bezeichner
zuweisungen	umgebung	e	wie oben
zuweisung	umgebung	e	wie oben
ausdruck	umgebung typ	e s	wie oben *string* oder *fixed* je nach Typ des Ausdrucks

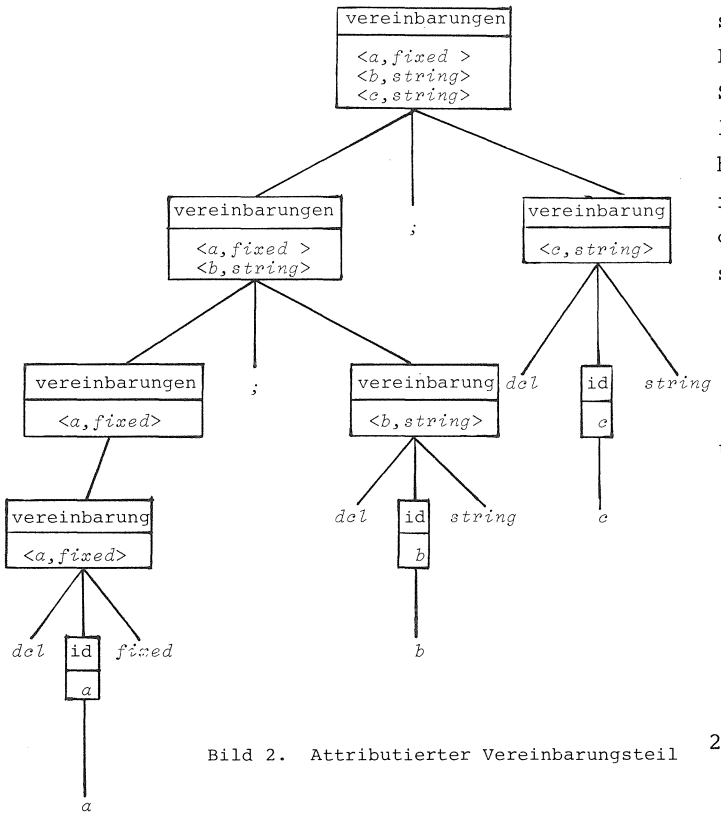

Bild 2. Attributierter Vereinbarungsteil

Im Anweisungsteil braucht gleichfalls fast jede syntatkische Variable das umgebungs-Attribut, damit die Typen der Variablen festgestellt werden können. Diesmal ist das Attribut jedoch aus dem Vereinbarungsteil ererbt. Zur Überprüfung der Typverträglichkeit der linken und rechten Seiten von Zuweisungen geben wir der syntaktischen Variablen ausdruck zusätzlich das synthetisierte Attribut typ (vgl. auch Tafel 2). B i l d 3 zeigt den attributierten Strukturbaum des Anweisungsteils aus dem laufenden Beispiel. T a f e l 4 enthält die zugehörigen Attributauswertungsregeln. Die dort benutzte Funktion TYP ist geeignet zu ergänzen. Für sie soll zum Beispiel gelten

$$\text{TYP}\ (b,\ \boxed{\begin{array}{l}\langle a,\ fixed\rangle\\ \langle b,\ string\rangle\\ \langle c,\ string\rangle\end{array}}\ ) = string$$

und

$$\text{TYP}\ (d,\ \boxed{\begin{array}{l}\langle a,\ fixed\rangle\\ \langle b,\ string\rangle\\ \langle c,\ string\rangle\end{array}}\ ) \text{ ist undefiniert.}$$

2.3. Implementierungsorientierte Attributierung

Der Attributauswertung stellen sich im allgemeinen einige Hindernisse entgegen. Meistens gibt es keine einfach zu ermittelnde Reihenfolge zur Berechnung der Attribute, die für alle Strukturbäume einer Grammatik gilt, und oft wird dadurch während der Berechnung der Strukturbaum in seiner Gesamtheit benötigt. Das trifft zwar in unserem einfachen Beispiel nicht zu, sollte aber für den allgemeinen Fall nicht überraschen.

Tafel 3. Attributierung des Vereinbarungsteils.
AR steht vor einer Attributauswertungsregel.
B steht vor einer Bedingung, die bei der Attributauswertung erfüllt sein muß. Zu den Produktionsregeln für die syntaktische Variable id sind geeignete Attributauswertungsregeln zu ergänzen.

```
vereinbarungen ::= vereinbarung
       AR:  vereinbarungen.umgebung := vereinbarung.eintrag

vereinbarungen ::= vereinbarungen ; vereinbarung
       AR:  vereinbarungen[1].umgebung :=
                  vereinbarungen[2].umgebung + vereinbarung.eintrag
        B:  ENTHALTEN(vereinbarung.eintrag.bezeichner,
                  vereinbarungen[2].umgebung) = false

vereinbarung   ::= dcl id fixed
       AR:  vereinbarung.eintrag := <id.bezeichner,fixed>

vereinbarung   ::= dcl id string
       AR:  vereinbarung.eintrag := <id.bezeichner,string>
```

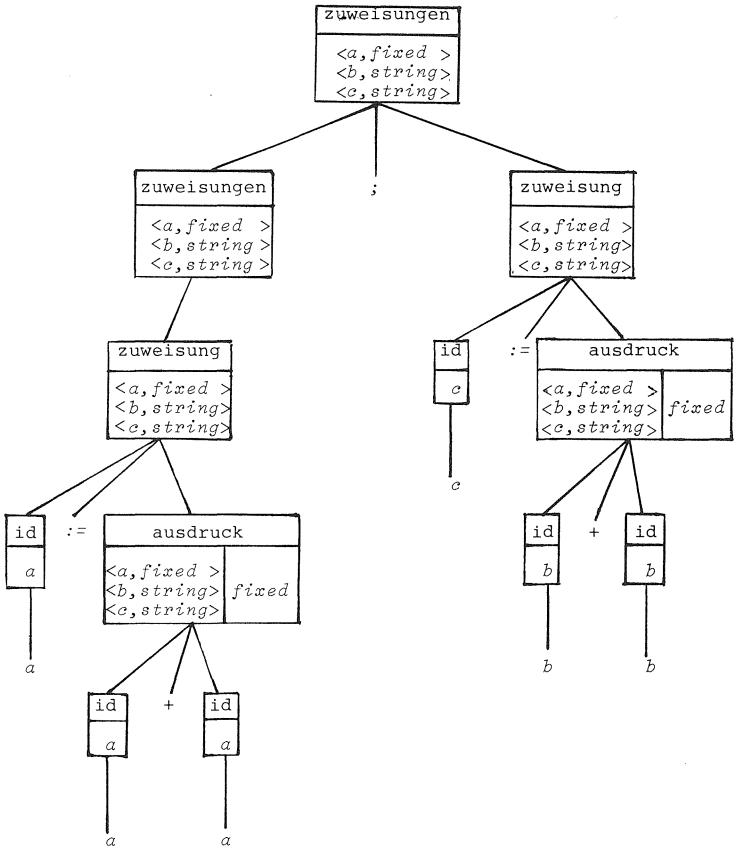

Bild 3. Attributierter Anweisungsteil.
Man beachte, daß "ausdruck" zwei Attribute hat.

Strukturbäume sind groß. So hatte ein 7-zeiliges, triviales PEARL-Programm schon mehr als 140 Knoten. In attributierten Strukturbäumen sind außerdem die Knoten groß, was dann für praktische Zwecke die Speicherung attributierter Bäume verbietet. Es wird berichtet [6], daß die Attributauswertung bei einem 42-zeiligen EUCLID Programm etwa 90 K Worte zu je 48 Bit erforderte. Die Ursache für diesen Effekt kann man am laufenden Beispiel sehen. An fast jedem Knoten des Strukturbaums hängt die

Tafel 4. Attributierung des Anweisungsteils.
AR steht vor einer Attributauswertungsregel.
B steht vor einer Bedingung, die bei der Attributauswertung erfüllt sein muß.

```
programm   ::= begin  vereinbarungen ; zuweisungen  end
       AR:   zuweisungen.umgebung := vereinbarungen.umgebung

zuweisungen  ::= zuweisung
       AR:   zuweisung.umgebung := zuweisungen.umgebung

zuweisungen  ::= zuweisungen  ;  zuweisung
       AR:   zuweisungen[2].umgebung := zuweisungen[1].umgebung
       AR:   zuweisung.umgebung := zuweisungen[1].umgebung

zuweisung    ::= id := ausdruck
       AR:   ausdruck.umgebung := zuweisung.umgebung
        B:   TYP(id.bezeichner,zuweisung.umgebung) =
                        ausdruck.typ

ausdruck ::= id + id
       AR:   ausdruck.typ := TYP(id[1].bezeichner,
                        ausdruck.umgebung)
        B:   TYP(id[1].bezeichner,ausdruck.umgebung) =
             TYP(id[2].bezeichner,ausdruck.umgebung)
```

Symboltabelle des ganzen Programms in Gestalt des Attributs umgebung. Neben dem gewaltigen Platzbedarf ergibt sich dadurch auch ein enormer Zeitbedarf, da zur Attributauswertung große Datenmengen bewegt werden müssen. Die zur Definition von Programmiersprachen naheliegende Art der Attributierung muß also verfeinert werden, ehe sie in praktischen Implementierungen benutzt werden kann. Dann allerdings ist sie ein unentbehrliches Hilfsmittel des Übersetzerbaus.

Im ersten Schritt zur Verbesserung der Effizienz wird eine günstige Berechnungsstrategie für die Attribute durch Bedingungen an die Attributauswertungsregeln erreicht. In praktischen Fällen wird oft erzwungen, daß die Attribute in einem links-rechts Durchlauf berechnet werden können (vgl. auch [7]). Gespeichert werden muß dabei nur ein Keller von Knoten in der Größenordnung eines kurzen Weges im Strukturbaum. Im laufenden Beispiel ergibt sich eine Berechnungsstrategie, die in B i l d 4 dargestellt ist.

Im zweiten Schritt zur Verbesserung der Effizienz wird die Verwendung globaler Größen in den Attributauswertungsregeln erlaubt. Damit wirken die Attributauswertungsregeln nicht mehr nur lokal in der Umgebung jeweils eines Knotens, sondern können auch Seiteneffekte auslösen. Im laufenden Beispiel machen wir die Symboltabelle zu einer globalen Größe. Dadurch verschwindet sie als Attribut aus dem Strukturbaum, ist nur noch in einem Exemplar vorhanden und braucht auch nicht mehr kopiert werden. Nur noch das typ- und das bezeichner-Attribut bleiben übrig. Die Strukturbäume werden erheblich ausgedünnt (B i l d 5). Natürlich stellen sich mit den globalen Größen die aus der Programmierung leidlich bekannten Schwierigkeiten mit den Seiteneffekten ein. Jedenfalls setzen sie eine abgestimmte Auswertungsstrategie voraus. So dürfen bei globaler Symboltabelle in der Beispielgrammatik die Attribute des Anweisungsteils erst nach den Attributen des Vereinbarungsteils abgearbeitet werden.

Für die Beispielgrammatik wird angenommen, daß die Symboltabelle mit den folgenden drei Prozeduren verwaltet wird.

```
TYP(X) = TYP des Bezeichners X;

ENTHALTEN(X) = true, wenn der
    Bezeichner X in der Symbol-
    tabelle eingetragen ist;
```

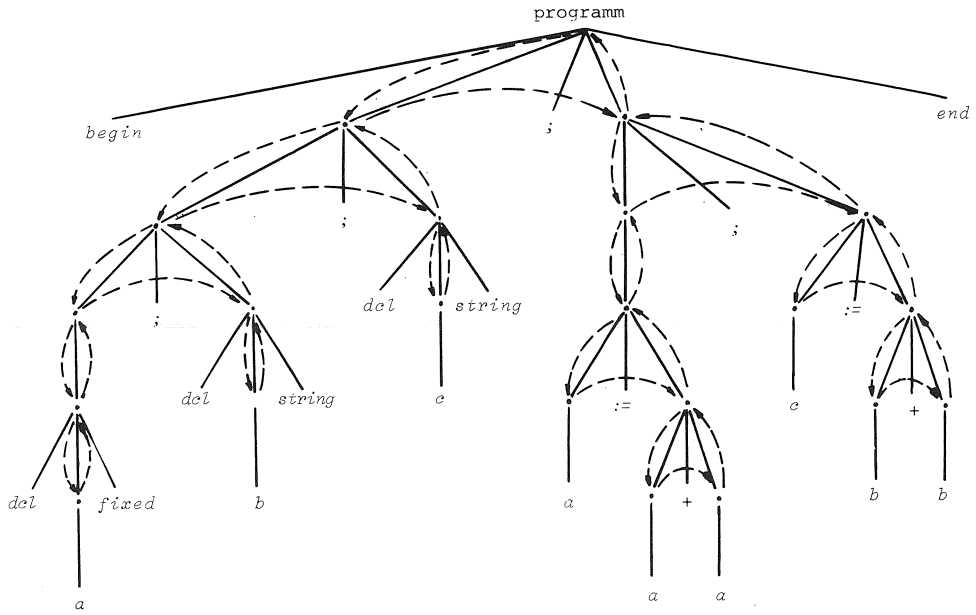

Bild 4. Berechnungsstrategie für die Attribute der Beispielgrammatik.
Die Attribute können bei einem Durchlauf des Strukturbaumes entlang der Pfeile berechnet werden. Jeder Knoten wird dabei zweimal besucht. Beim ersten Mal werden seine ererbten, beim zweiten Mal seine synthetisierten Attribute berechnet.

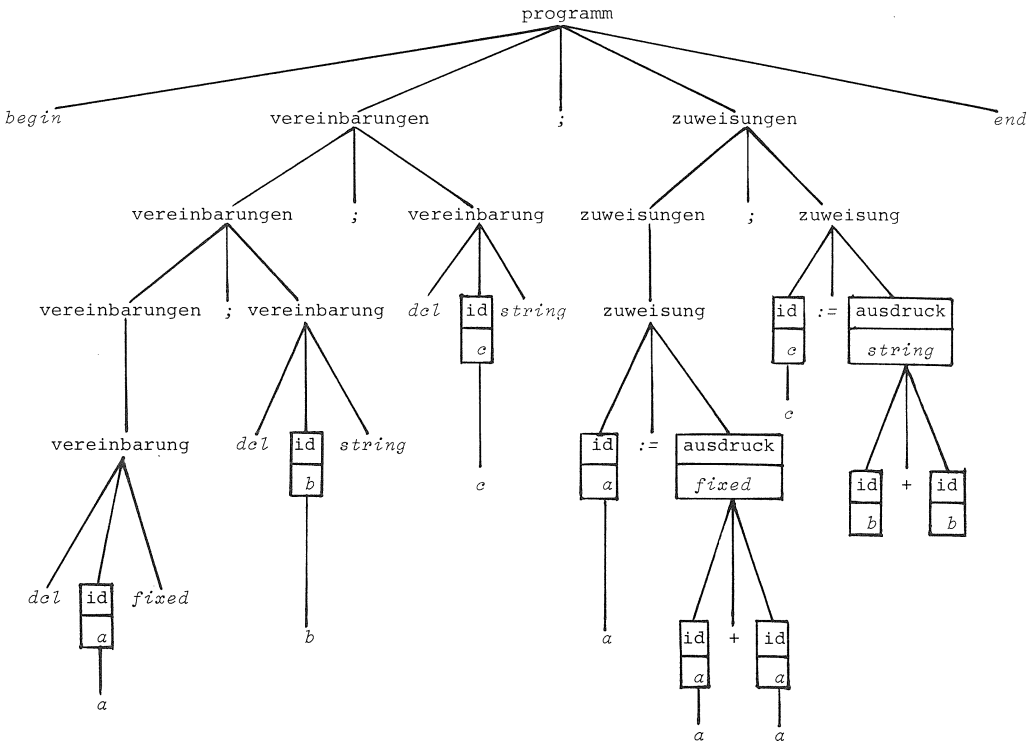

Bild 5. Attributierter Strukturbaum bei implementierungsorientierter Attributierung und globalen Attributen.

EINTRAGE(X,Y) trägt den Bezeichner X mit dem Typ Y in die Symboltabelle ein.

T a f e l 5 enthält die so attributierte Grammatik.

3. Folgerungen

Die vorstehenden Beispiele zeigen, daß einer der für attributierte Grammatiken als Definitionsmethode ins Feld geführten Vorteile eine Finte ist. Zwar läßt sich auch

Tafel 5. Implementierungsorientierte Attributierung. Geeignete Attributauswertungsregeln für die syntaktische Variable id werden unterstellt.

```
programm        ::= begin  vereinbarungen ; zuweisungen  end

vereinbarungen  ::= vereinbarungen ; vereinbarung

vereinbarungen  ::= vereinbarung

vereinbarung    ::= dcl id fixed
        AR:     if ENTHALTEN(id.bezeichner)
                   then call FEHLER(...)
                   else call EINTRAG(id.bezeichner,fixed)

vereinbarung    ::= dcl id string
        AR:     if ENTHALTEN(id.bezeichner)
                   then call FEHLER(...)
                   else call EINTRAG(id.bezeichner,string)

zuweisungen     ::= zuweisungen ; zuweisung

zuweisungen     ::= zuweisung

zuweisung       ::= id := ausdruck
        AR:     if TYP(id.bezeichner) ≠ ausdruck.typ
                   then call FEHLER(...)

ausdruck        ::= id + id
        AR:     if   ¬ENTAHLTEN(id[1].bezeichner) ∨
                     ¬ENTHALTEN(id[2].bezeichner) ∨
                      TYP(id[1].bezeichner) ≠ TYP(id[2].bezeichner)
                then call FEHLER(...)
                ausdruck.typ := TYP(id[1].bezeichner)
```

die attributierte Grammatik der PEARL Norm einem Übersetzer erzeugenden System ("compiler compiler") eingeben, doch entsteht dadurch kein Übersetzer im praktischen Sinne. Dennoch ergibt sich ein nicht zu unterschätzender Vorteil, indem dadurch die Zyklenfreiheit der Grammatik bewiesen wird und indem Fehler ausgemerzt werden, die zu Verstößen gegen die formalen Bildungsgesetze attributierter Grammatik führen (vgl. auch [3]).

Zur Lösung eines anderen wichtigen Problems können leider auch solche Systeme nicht beitragen: Definiert die Grammatik die beabsichtigte Sprache? Dieses Problem ist im Software Engineering wohl bekannt und wird natürlich nicht mit einem Schlagwort gelöst. Ich werde dadurch jedoch an die Forderung der Lesbarkeit erinnert und muß feststellen, daß die Norm zwar lesbarer sein mag als andere halbformale Sprachdefinitionen, daß sie aber nicht lesbar genug ist, um zu einem Verkaufsschlager zu werden, der die augenblickliche Geldnot des NI im DIN lindert.

Literatur

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

[2] Heilbrunner, S., Schmitz, L.: Zur attributierten Grammatik von PEARL. PEARL-Rundschau, Heft 3, Band 2 (1981) S. 7 - 17.

[3] Kastens, U., Zimmermann, F.: GAG - A Generator based on attributed grammars. Institut für Informatik II, Universität Karlsruhe, Bericht 14/80 (1980).

[4] Lewis, P.M., Rosenkrantz, D.J., Stearns, R.E.: Compiler Design Theory, Reading, Mass.: Addison-Wesley (1976).

[5] Marcotty, M., Ledgard, H.F., Bochmann, G.V.: A Sampler of Formal Definitions. Computing Surveys Vol. 8, No. 2 (1976) S. 191 - 276.

[6] Räihä, K.-J.: Experiences with the compiler writing system HLP. In: Semantics-Directed Compiler Generation, Proceedings of a Workshop in Aarhus (N.D. Jones, ed.) Lecture Notes in Computer Science, vol. 94, Springer (1981), S. 350 - 362.

[7] Wilhelm, R.: Attributierte Grammatiken. Informatik-Spektrum Bd. 2, No. 3 (1979) S. 123 - 130.