Lecture Notes in Computer Science

Edited by G. Goos, Karlsruhe and J. Hartmanis, Ithaca

7

GI
Gesellschaft für Informatik e.V.

3. Fachtagung über Programmiersprachen

Kiel, 5.–7. März 1974

Springer-Verlag
Berlin · Heidelberg · New York

Lecture Notes in Computer Science

Edited by G. Goos, Karlsruhe and J. Hartmanis, Ithaca

7

GI

Gesellschaft für Informatik e.V.

3. Fachtagung über Programmiersprachen

Kiel, 5.–7. März 1974

Herausgegeben von Bodo Schlender und Wolfgang Frielinghaus

Springer-Verlag
Berlin · Heidelberg · New York 1974

Professor Dr. Bodo Schlender
Institut für Informatik und
Praktische Mathematik
Universität Kiel

Wolfgang Frielinghaus
Telefunken GmbH Konstanz

AMS Subject Classifications (1970): 00A10, 02G05, 68-02, 68A05, 68A10, 68A20, 68A30

ISBN 3-540-06666-7 Springer-Verlag Berlin · Heidelberg · New York
ISBN 0-387-06666-7 Springer-Verlag New York · Heidelberg · Berlin

Offsetprinting and bookbinding: Julius Beltz, Hemsbach/Bergstr.

Vorwort

Unter den Fachtagungen der Gesellschaft für Informatik ist dies die dritte, die speziell den Programmiersprachen gewidmet ist. Die Vorgängertagungen fanden im März 1971 in München*) und im März 1972 in Saarbrücken**) statt.

Während die Jahrestagungen der GI versuchen, einen Überblick über alle wesentlichen Bereiche der Informatik und einen wissenschaftlichen Gedankenaustausch zwischen verschiedenen Fachgebieten zu vermitteln, ist es der Zweck der Fachtagungen, den wissenschaftlichen Fortschritt in einem Spezialgebiet in Vortrag und Diskussion der Öffentlichkeit darzustellen.

Zur Entwicklung des Fachgebietes ist es einmal notwendig, in regelmäßigen Zeitabständen Gelegenheit zu geben, über die letzten wissenschaftlichen Forschungsergebnisse zu berichten, zum anderen müssen Arbeiten aus den Anwendungen vorgestellt werden können, um die erforderliche gegenseitige Anregung von Theorie und Praxis Wirklichkeit werden zu lassen.

Um eine vernünftige Gliederung der Tagung zu ermöglichen, ist es zweckmäßig, daß der Programmausschuß in der Einladung zur Anmeldung von Vorträgen spezielle Schwerpunkte setzt. Für die gegenwärtige Fachtagung waren dies die Gebiete Semantik von Programmiersprachen, Übersetzerbau, Dialogsprachen und Programmiersprachen für Prozeßrechner. Der Programmausschuß war bei der Zusammenstellung des Tagungsprogrammes bemüht, diejenigen Vorträge auszuwählen, die den oben genannten Zielen einer Fachtagung am besten entsprachen. Überdies war es möglich, eine Reihe von international bekannten Wissenschaftlern zu einführenden Hauptvorträgen zu gewinnen.

---

*) Lecture Notes in Economics and Mathematical Systems, Band 75, Springer-Verlag, Berlin . Heidelberg . New York 1972

**) Gesellschaft für Informatik e.V., Bericht Nr. 4, veröffentlicht durch die Gesellschaft für Mathematik und Datenverarbeitung mbH, 5205 St. Augustin 1, 1972

Dem Programmausschuß gehörten an:

Prof. Dr. K. Alber, Braunschweig
W. Frielinghaus, Konstanz
Prof. Dr. H. Langmaack, Saarbrücken
Prof. Dr. M. Paul, München
Prof. Dr. B. Schlender, Kiel
Prof. Dr. G. Seegmüller, München.

Allen, die am Zustandekommen der Tagung mitgewirkt oder ihre Durchführung unterstützt haben, möchten wir an dieser Stelle Dank sagen, besonders

den Vortragenden, den Sitzungsleitern und den Diskussionsrednern,
den Mitgliedern des Programmausschusses,
dem Bundesministerium für Forschung und Technologie und
der Universität Kiel, insbesondere dem Institut für Informatik und Praktische Mathematik.

Besonderer Dank gebührt dem Springer-Verlag, der die Veröffentlichung dieses Tagungsbandes in so kurzer Frist ermöglichte, so daß er noch zu Beginn der Tagung bereitgestellt werden konnte.

Frau G. Rasch, Kiel, sei an dieser Stelle für ihre Sekretariatsarbeiten, die meist unter Zeitdruck abgewickelt werden mußten, herzlich gedankt.

Wir wünschen allen Tagungsteilnehmern einen anregenden Verlauf der Tagung und einen angenehmen Aufenthalt in Kiel.

Kiel, im Januar 1974

W. Frielinghaus
B. Schlender

Inhaltsverzeichnis

HAUPTVORTRAG:

Automatic Programming
Z. Manna ..... *)

Zum Begriff der Modularität von Programmiersprachen
H. Langmaack 1

Entscheidbarkeitsprobleme bei der Übersetzung von Programmen mit einparametrigen Prozeduren
W. Lippe 13

Eine axiomatische Studie zur Identifikation von Identifikatoren
F. Kröger 25

Zur Übersetzbarkeit von Programmiersprachen
H. Jürgensen 34

Über die Syntax von Dialogsprachen
I. Kupka, N. Wilsing 45

Definition und Implementierung eines Dialogsystems
H. Rohlfing 55

A Project Oriented Support Library
R.W. Friday 68

HAUPTVORTRAG:

Feedback-free Modularization of Compilers
W.M. McKeeman, F.L. DeRemer 78

Zur Minimalität des Hilfszellenbedarfs von systematisch übersetzten Ausdrücken
U. Peters 89

A PASCAL Compiler bootstrapped on a DEC-System 10
H.-H. Nagel et al. 101

*) Manuskript konnte vom Verfasser nicht fertiggestellt werden.

Erfahrungen mit einem Parser-Generator für Chomsky-Grammatiken

R. Nollmann 114

HAUPTVORTRAG:

Structured Programs, Arcadian Machines and the Burroughs B1700

W.T. Wilner 133

Struktureinheiten in kontextfreien Sprachen und ihre Übersetzung

W. Niegel 149

ALSI - eine höhere Programmiersprache zur Transformation von ALGOL 68 - Verbunden in SIMULA-Klassen

U. Kastens 162

Interaktives Trace- und Debugging-System ALGOL KIEL X 8

J. Baginski, H. Seiffert 173

HAUPTVORTRAG:

Some Characteristics of RTL/2 and their Relevance to Process Control

J.G.P. Barnes 189

Prozeßorientierte Programmiersprachen, Formen und Funktionen

G. Koch, V. Kussl 199

Ein Vergleich von Prozess-Kontroll-Sprachen als Basis für den Normungsvorschlag einer internationalen Prozeßrechnersprache

K.F. Kreuter 219

ZUM BEGRIFF DER MODULARITÄT VON PROGRAMMIERSPRACHEN

H. LANGMAACK

1. Modulares Programmieren

Die gegenwärtige Arbeit schließt an Überlegungen an, die J.B. Dennis in seiner Schrift "Modularity" [3] angestellt hat. Er hebt dort die folgenden Ziele modularen Programmierens hervor:
1. Man muß sich von der Korrektheit eines Programmmoduls unabhängig vom Kontext in größeren Softwareeinheiten überzeugen können.
2. Man muß Programmmodule ohne Kenntnis ihrer internen Arbeitsweisen zusammensetzen können.
Diese Ziele verkörpern auch den Begriff der Kontextunabhängigkeit bei W.E. Boebert [2] .

In ALGOL-artigen Programmiersprachen sind es offensichtlich die Prozeduren, die für die Betrachtung als Programmmodule in Frage kommen. Dabei muß man sich aber auf diejenigen Prozeduren beschränken, die keine nicht lokalen (globalen) Parameter haben, da ihre Verwendung in einem Programmmodul das Konzept der Modularität verletzen würde. Lediglich das Auftreten globaler Standardidentifikatoren darf erlaubt sein, die wir der Einfachheit halber zu den Grundgrößen der Programmiersprache hinzuzählen.

Wir nennen ein Programm modular, wenn in ihm nur Prozeduren ohne globale Parameter Verwendung finden. Nach Dennis ist Modularität aber nicht bloß eine Eigenschaft einzelner Programme, vielmehr sollte Modularität vor allem als Eigenschaft einer Programmiersprache selbst angesehen werden. Für Programmiersprachen erhebt sich nämlich die Frage, ob jedes Programm in ein äquivalentes modulares Programm der gleichen Sprache

verwandelt werden kann. Wenn das zutrifft, dann wollen wir eine Programmiersprache modular nennen.

Bislang ist der Begriff der Modularität von Programmiersprachen noch nicht vollständig umrissen. Noch hängt er von einer präzisen Fassung des Begriffs Äquivalenz ab. Die funktionale Äquivalenz, bei der zwei Programme äquivalent heißen, wenn sie das gleiche Ein-Ausgabeverhalten zeigen, ist wenig sinnvoll. Denn sonst gäbe es zu jedem Programm einer ALGOL-artigen Programmiersprache ein äquivalentes prozedurloses, also modulares Programm, und jede ALGOL-artige Programmiersprache wäre trivialerweise modular.

Bei der Umwandlung eines Programms in ein äquivalentes modulares sollte die Prozedurstruktur im wesentlichen erhalten bleiben. Modularität sollte vornehmlich dadurch erzielt werden können, daß globale Parameter in lokale verwandelt werden. Versierte Programmierer kennen diesen Prozess sehr gut, und oft hört man die Meinung, die Verwandlung in lokale Parameter sei doch stets möglich. Auf dabei sich zeigende Probleme weist bereits Dennis in [3] hin, und wir werden sehen, daß dieser Verwandlung Grenzen gesetzt sind (Satz 3 und 4).

2. Formale Äquivalenz von Programmen

Um dem Erfordernis nach Erhaltung der Prozedurstruktur entgegenzukommen, führen wir den Begriff der formalen Äquivalenz ein. Dabei stützen wir uns auf die Programmiersprache ALGOL 60, die wir zum Zwecke präziser Formulierungen zu ALGOL 60-P nur unwesentlich modifizieren. ALGOL 60-P ist gegenüber ALGOL 60 in folgender Weise abgewandelt [5] :

a) Es sind nur eigentliche Prozeduren, keine Funktionsprozeduren zugelassen.

b) Wertaufruflisten und Spezifikationen in Prozedurdeklarationen fehlen.

c) Es sind nur Identifikatoren als aktuelle Parameter in Prozeduranweisungen erlaubt.

d) Neben begin und end haben wir ein zusätzliches Paar von Anweisungsklammern { }. Sie wirken wie Block-begin und Block-end. Wir verlangen, daß alle Prozedurrümpfe in diese Klammern eingeschlossen sind, und wir nennen sie in diesem Kontext Rumpfklammern. In anderem Kontext heißen sie Aufrufklammern. Wir benötigen diese Klammern, um originale Programme, die der Programmierer geschrieben hat, von Programmen zu unterscheiden, die durch Anwenden der Kopierregel entstanden sind.

Es dürfte klar sein, was ein syntaktisches ALGOL 60-P-Programm Π ist. Π heißt darüber hinaus formal, wenn zu jedem Vorkommen eines Identifikators genau ein definierendes Vorkommen gehört. Formale Programme, die durch zulässige Umbenennung von Identifikatoren auseinander hervorgehen, sehen wir als identisch an. O.B.d.A. darf jedes formale Programm als ausgezeichnet angenommen werden, d.h. verschiedene definierende Vorkommen von Identifikatoren sind durch verschiedene Identifikatoren benannt. Ein formales Programm heißt kompilierbar (oder eigentlich wie im ALGOL 68-Bericht [7]), wenn jeder Identifikator gemäß seines definierenden Vorkommens vernünftig angewandt ist. Kompilierbare Programme sind diejenigen, die von einem Kompiler akzeptiert und übersetzt werden können. Ein formales Programm Π heißt partiell kompilierbar, wenn nach Ersetzen aller Prozedurrümpfe $\{\varrho\}$ durch leere Rümpfe $\{\ \}$ das resultierende Programm Π_e kompilierbar ist.

Π sei partiell kompilierbares Programm, und Π' sei formal. Dann heißt es von Π', es ergebe sich aus Π durch Anwenden der Kopierregel ($\Pi \vdash\!\!\!- \Pi'$), wenn folgendes gilt: Sei $f(a_1,\ldots,a_n)$ eine Prozeduranweisung im Hauptprogramm von Π . Sei

$$\varphi = \underline{\text{proc}}\ f(x_1,\ldots,x_n)\ ;\ \{\varrho\};$$

die zugehörige Prozedurdeklaration φ . Die partielle Kompilierbarkeit von Π garantiert die gleiche Anzahl n der aktuellen und formalen Parameter. Wir dürfen Π als ausgezeichnet voraussetzen. Dann entsteht Π' aus Π dadurch, daß $f(a_1,\ldots,a_n)$ durch einen modifizierten Rumpf $\{\varrho_m\}$ von $\{\varrho\}$, einen erzeugten Block, ersetzt wird, wobei die formalen Parameter x_i in $\{\varrho\}$ durch die entsprechenden aktuellen Parameter a_i ausgetauscht werden. Dabei sind die Rumpfklammern $\{\ \}$ um ϱ in Π zu Aufrufklammern $\{\ \}$ um ϱ_m in Π' geworden.

Π = ... proc $f(x_1,\ldots,x_n)$; $\{\varrho\}$;...; $f(a_1,\ldots,a_n)$;...

Π' = ... proc $f(x_1,\ldots,x_n)$; $\{\varrho\}$;...; $\{\varrho_m\}$;...

$\vdash\!\!\!-$, $\vdash\!\!\!-^{+}$ und $\vdash\!\!\!-^{*}$ sind Relationen in der Menge der formalen Programme. Ein formales Programm Π heißt original, wenn die Anweisungsklammern $\{\ \}$ in Π nur als Rumpfklammern verwendet sind. Für ein originales Programm Π heißt die Menge

$$E_\Pi := \{\Pi' \mid \Pi \vdash\!\!\!-^{*} \Pi'\}$$

die Ausführung von Π. Der Ausführungsbaum T_Π von Π besteht aus denjenigen Programmen in E_Π, die höchstens ein innerstes Aufrufklammernpaar

besitzen. Jedes Programm (jeder Knoten) Π'' in T_Π besitzt höchstens einen unmittelbaren Vorgänger $\Pi' \longmapsto \Pi''$ in E_Π.

Um zum Begriff der formalen Äquivalenz von Programmen zu gelangen, führen wir den Begriff der reduzierten Ausführung ein. Wir bilden für jedes Programm $\Pi' \in E_\Pi$ das zugehörige Hauptprogramm Π'_m durch Streichen aller Prozedurdeklarationen, wir ersetzen jede verbliebene Prozeduranweisung in Π'_m durch ein Spezialsymbol, z.B. call, und wir nennen das Ergebnis das reduzierte Hauptprogramm Π'_r von Π'.

$$E_{r\Pi} := \{\Pi'_r | \Pi' \in E_\Pi\}$$

heißt die reduzierte Ausführung,

$$T_{r\Pi} := \{\Pi'_r | \Pi' \in T_\Pi\}$$

der reduzierte Ausführungsbaum von Π. Zwei originale Programme heißen nun formal äquivalent, wenn ihre reduzierten Ausführungen bzw. Ausführungsbäume identisch sind. Damit ist die formale Äquivalenz zweier Programme so eingeführt worden, daß sie im wesentlichen gleiche Prozedurstrukturen haben. Ferner gilt der

Satz 1: Formal äquivalente Programme sind auch funktional äquivalent.

Von einem originalen Programm Π heißt es, es habe formal korrekte Parameterübergaben, wenn alle Programme $\Pi' \in E_\Pi$ bzw. T_Π partiell kompilierbar sind. Π heißt darüber hinaus Makroprogramm, wenn E_Π bzw. T_Π endlich ist. Makroprogramme sind diejenigen, die mit Makro- oder offener Unterprogrammtechnik implementiert werden können. Es gibt keinen Algorithmus, der für gegebenes originales Programm Π entscheidet, ob Π formal korrekte Parameterübergaben hat bzw. ob Π Makroprogramm ist. Wir haben jedoch den

Satz 2: Formale Äquivalenz von Programmen ist invariant gegenüber den beiden Eigenschaften, formal korrekte Parameterübergaben zu haben bzw. Makroprogramm zu sein.

3. Die Nichtmodularität von ALGOL 60-P

Wir wollen nun der Frage nach der Modularität von ALGOL 60-P nachgehen. Diese Frage hängt sehr eng mit dem Entscheidungsproblem für formal korrekte Parameterübergaben zusammen. Für ALGOL 60-P-Programme Π ohne globale formale Parameter, d.h. für die Teilsprache ALGOL 60-P-0, ist nämlich entscheidbar, ob Π formal korrekte Parameterübergaben hat. Wenn wir nun ein effektives Verfahren hätten, das jedes originale Programm Π in ein formal äquivalentes modulares verwandelte, dann hätten wir

auch eine positive Lösung des genannten Entscheidungsproblems für ALGOL 60-P. Daher gilt der manchen Programmierer überraschende

Satz 3: Es gibt kein effektives Verfahren, das jedes originale ALGOL-60-P-Programm in ein formal äquivalentes modulares der gleichen Sprache verwandelt.

Dieser Satz läßt zwar die Nichtmodularität von ALGOL 60-P vermuten, beweist sie jedoch noch nicht. Dazu müssen wir erst ein Programmbeispiel konstruieren, das zu keinem modularen Programm formal äquivalent ist:

```
Π¹ = begin proc M(x,y);{ x(y)};
          proc E(η);  {      };
          proc L(x,y);
              {proc A(η); {η(x,M)};
               proc Ā(ξ,α);{α(ξ,y)};
               L(A,Ā);M(A,Ā)};
          L(E,M) end
```

```
...{...L(A',Ā');M(A',Ā')}...
          ...{A'(Ā')}...                    }
          ...{      }...                    }  5 Knoten,
...{...L(A^(ν),Ā^(ν);M(A^(ν),Ā^(ν))}...     }  siehe unten
                                            }  ν ≥ 2
          ...{A^(ν)(Ā^(ν))}...              }
          ...{Ā^(ν)(A^(ν-1),M)}...          }
          ...{M(A^(ν-1),Ā^(ν-1))}...        }
                                            }
          ...{M(A',Ā')}...                  }
          ...{A'(Ā')}...                    }
          ...{Ā'(E,M)}...                   }  3 · ν
          ...{M(E,M)}...                    }  Knoten
          ...{E(M)}...                      }
          ...{      }...                    }
```

Die Identifikatoren $A^{(\nu)}$, $\bar{A}^{(\nu)}$ bezeichnen modifizierte Kopien der Prozeduren A, $\bar{A}$. Der Ausführungsbaum T_{π^1} ist offenbar unendlich, und zu jeder natürlichen Zahl ν existiert ein Knoten, z.B. $\ldots\{M(A^{(\nu)},\bar{A}^{(\nu)})\}\ldots$, derart daß der aufsitzende Teilbaum endlich ist und eine Tiefe $>\nu$ besitzt. Dann aber kann π^1 zu keinem modularen Programm formal äquivalent sein.

Satz 4: ALGOL 60-P ist nicht modular.

Dagegen gilt

Satz 5: ALGOL 60-P-O ist modular; es gibt ein effektives Verfahren, das jedes originale Programm in ein formal äquivalentes modulares verwandelt.

Das Verfahren werde am Beispiel π^2 demonstriert, um dort die globalen Parameter a und p der Prozedur q zu eliminieren:

```
π² = begin proc p(x);
                {real a;
                 proc q(y); {a:= 2+a;p(q);y(p);q(y)};
                 x(q);p(x)};
          p(p)  end
```

Im transformierten Programm π^2_t haben die Prozeduren p und q zusätzliche formale Begleitparameter $\bar{p},\bar{\bar{p}},\bar{x},\bar{\bar{x}},\bar{q},\bar{\bar{q}},\bar{y},\bar{\bar{y}}$ erhalten:

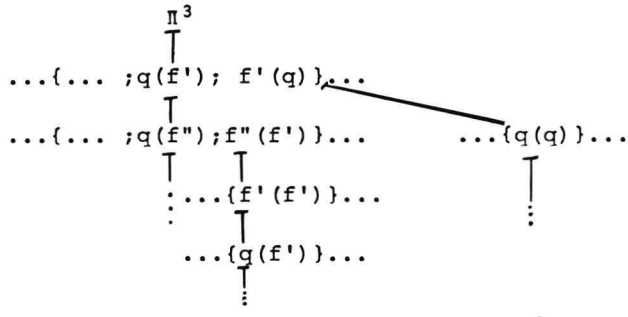
```
π²t = begin proc p(p̄,p̿,x,x̄,x̿);
                {real a;
                 proc q(q̄,q̿,y,ȳ,ÿ);
                     {q̄ := 2+q̄; q̿( , ,q,q̄,q̿);
                      y(ȳ,ÿ,q̄, , ); q(q̄,q̿,y,ȳ,ÿ)};
                 x(x̄,x̿,q,a,p); p( , ,x,x̄,x̿)};
           p( , ,p, , ) end
```

In Prozeduranweisungen erhalten formale Identifikatoren x und y die aktuellen Begleitparameter $\bar{x},\bar{\bar{x}},\bar{y},\bar{\bar{y}}$. q erhält die aktuellen Begleitparameter a,p. Die aktuellen Begleitparameter anderer nicht formaler Identifikatoren sind unwichtig und können irgendwelche lokale Identifikatoren sein. Anschließend werden im Rumpf von q a,p durch $\bar{q},\bar{\bar{q}}$ ersetzt. Wer auf dem Standpunkt steht, selbst q sei globaler Parameter von q kann auch diesen Parameter in einen lokalen formalen verwandeln. Für das

Programmbeispiel

```
π³ = begin proc q(r);
                {proc f(x);{r(x)};
                 q(f);f(r)};
             q(q) end
```

mit dem Aufrufbaum T_{π^3}



und dem globalen formalen Parameter r von f hätte ein analoges Verfahren unendlich viele Begleitparameter benötigt. Die Identifikatoren f',f",... bezeichnen modifizierte Kopien der Prozedur f.

4. Eine modulare Obersprache von ALGOL 60-P

Modulare Programme haben gegenüber anderen den Vorteil, daß man sie formal äquivalent transformieren kann, indem man ihre Prozeduren herausnimmt und der Reihe nach unmittelbar hinter das erste begin setzt. Auf diese Weise wird vermieden, daß die Kopierregel neue Prozedurdeklarationen mit neuen Prozeduridentifikatoren schafft, die die Einsicht in das Wirken der Kopierregel empfindlich stören. Daher rührt der Wunsch, ALGOL 60-P zu einer modularen Programmiersprache ALGOL 60-P-G zu erweitern.

Prozedurdeklarationen in ALGOL 60-P-G haben die Form

(*) proc f<$y_1,\ldots,y_{m_f}$> ($x_1,\ldots x_{n_f}$);{ $\wp$ };.

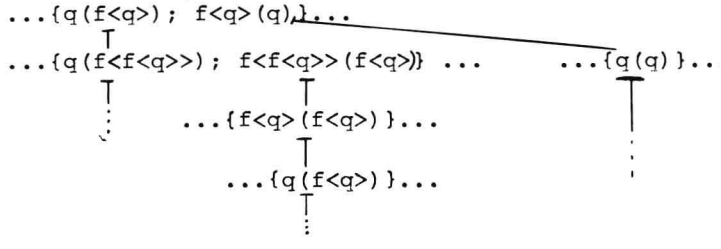
Die Identifikatoren $y_1,\ldots,y_{m_f}$ heißen formale Parameter neuer Art, $x_1,\ldots,x_{n_f}$ sind die formalen Parameter alter Art. Wenn m_f oder n_f Null sind, schreiben wir die spitzen bzw. runden Klammern nicht. Alle übrigen Deklarationen sehen wie in ALGOL 60-P aus.

Ein Term ist eine endliche Zeichenreihe, die nach folgenden Regeln gebildet wird:

1. Identifikatoren sind Terme.
2. Wenn ψ Identifikator ist und $\tau_1,\ldots,\tau_m$, $m \geq 1$, Terme sind, dann ist auch $\psi\langle\tau_1,\ldots,\tau_m\rangle$ ein (echter) Term. τ_i ist aktueller Parameter neuer Art.

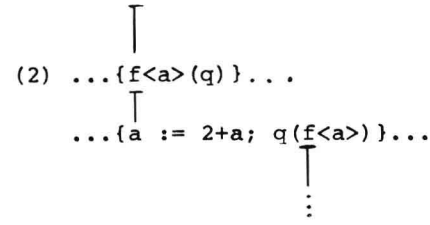
Ein syntaktisches ALGOL 60-P-G-Programm Π entsteht nun aus einem syntaktischen ALGOL 60-P-Programm Π' dadurch, daß Prozedurköpfe in Π' durch neue Köpfe der obigen Form (*) ersetzt werden und daß man in Π' angewandt vorkommende Identifikatoren durch Terme auswechselt. Auch alle übrigen Begriffe lassen sich ohne Schwierigkeit auf ALGOL 60-P-G ausdehnen. Beispiel für das Wirken der Kopierregel:

```
Π⁴ = begin proc f<f̄>(r); {f̄(x)};
           proc q(r);{q(f<r>); f<r>(r)};
           q(q) end
```



Satz 1 und 2 gelten auch für ALGOL 60-P-G. Die Sprachen ALGOL 60-P und ALGOL 60-P-G sind nur unwesentlich verschieden, wenn man sich auf Programme ohne formale Prozeduranweisungen beschränkt. Der Unterschied liegt in der Methode wie man Prozeduren f, die durch formale Prozeduranweisungen (1) aufgerufen werden, mit aktuellen Parametern versorgt. In ALGOL 60-P werden alle aktuellen Parameter im Moment des Aufrufs übergeben. In ALGOL 60-P-G werden die aktuellen Parameter alter Art ebenfalls im Moment des Aufrufs übergeben, während die aktuellen Parameter neuer Art (2) so betrachtet werden können, als ob sie bereits vorher übergeben worden wären, als der Prozeduridentifikator f als aktueller Parameter eines vorangegangenen Aufrufs (3) auftrat. Beispiel:

```
Π⁵ = begin real a; proc q(y);{y(q)};   (1)
           proc f<r>(x);{r := 2+r; x(f<r>)};
      (3)  q(f<a>) end
```



Aus der Wirksamkeit der Parameter neuer Art rührt der

Satz 6: ALGOL 60-P-G ist modular; es gibt ein effektives Verfahren, das jedes originale Programm in ein formal äquivalentes modulares umwandelt.

Die Anwendung dieses Verfahrens auf Programm Π^3 liefert

```
Π³t = begin proc q(r);
                {proc f<f̄>(x);{f̄(x)};
                 q(f<r>); f<r>(r)};
      q(q) end
```

Hier bekommt nur die Prozedur f einen formalen Begleitparameter $\bar{f}$ neuer Art. Der aktuelle Begleitparameter neuer Art von f ist r. Anschließend wird im Rumpf von f r durch $\bar{f}$ ersetzt. Π^3_t und Π^4 sind offensichtlich formal äquivalent.

Satz 7: Die in Satz 5 und 6 genannten Verfahren lassen die Eigenschaften einer Prozedur, formal erreichbar bzw. formal rekursiv zu sein, invariant.

Dabei heißt eine Prozedur φ in Π formal erreichbar, falls ein Programm Π' in T_Π existiert, dessen innerster erzeugter Block modifizierter Rumpf einer (eventuell modifizierten) Kopie von φ ist. φ heißt formal rekursiv, falls Programme $\Pi' \overset{+}{\longmapsto} \Pi''$ in T_Π existieren, deren innerste erzeugte Blöcke modifizierte Rümpfe von Kopien von φ sind.

Erst die Einbettung von ALGOL 60-P in die modulare Sprache ALGOL 60-P-G hat einen durchsichtigen Beweis für die algorithmische Unlösbarkeit des Makroprogrammproblems für ALGOL 60-P geliefert [6] . Wegen Satz 2 genügt es nämlich, in ALGOL 60-P-G Programme ohne Prozedurschachtelungen zu betrachten.

Für die Teilsprache ALGOL 60-P-F von ALGOL 60-P-G derjenigen Programme ohne Parameter alter Art gilt aufgrund des in Satz 6 genannten Verfah-

rens

Satz 8: ALGOL 60-P-F ist modular.

Die ALGOL 60-P-F-Programme ohne Prozedurschachtelungen können mit den Makrogrammatiken von M.J. Fischer [4] identifiziert werden. Deshalb sind das Problem der formal korrekten Parameterübergaben und das Makroprogrammproblem für ALGOL 60-P-F algorithmisch lösbar. Weil die beiden Sprachen ALGOL 60-P und ALGOL 60-P-F jeweils Randmengen von ALGOL 60-P-G darstellen, wird ihr gegensätzliches Verhalten verständlich.

5. Zur Implementierung von ALGOL 60-P-G

Nachteilig für die Implementierung von ALGOL 60-P-G ist zunächst, daß bei sukzessivem Anwenden der Kopierregel die aktuellen Parameter unvorhersehbar umfangreich werden können (siehe Π^4), so daß sie nicht mehr wie bei ALGOL 60-P in jeweils einer Zelle Platz finden. Selbst wenn ganze Ausdrücke wie in ALGOL 60 oder Anweisungen wie in ALGOL 68 als aktuelle Parameter zugelassen sind, kann man sich auf ALGOL 60-P zurückziehen, indem man "dicke" aktuelle Parameter zu Rümpfen parameterloser Prozeduren macht und die Parameter selbst durch die zugehörigen Prozeduridentifikatoren ersetzt. Dieser Weg führt bei ALGOL 60-P-G nicht weiter. Bei geeigneter Verweistechnik im Laufzeitkeller [1] kann man aber auch hier alle aktuellen Parameter in jeweils einer Zelle unterbringen.

Gewöhnliche und formale Prozeduranweisungen

$$f\ \langle b_1,\ldots,b_m\rangle(a_1,\ldots,a_n)$$

bzw.

$$x(a_1,\ldots,a_n)$$

seien auf folgende Weise in Maschinencode übersetzt:

```
UNT  Gew Proz Anw          bzw.   UNT  Form Proz Anw
       f                                  x
      <b1                                 (a1,
       :                                   :
Bi:    bi,                                 an);
       :
       bm>                          R:
      (a1
       :
       an);
R:
```