KfK 3239
Juni 1982

PL/I-FORMAC Programm (Precompiler) zur Einführung neuer Sprachmerkmale in FORTRAN IV

W. Abel
Institut für Angewandte Kernphysik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Angewandte Kernphysik

KfK 3239

PL/I-FORMAC Programm (Precompiler) zur Einführung neuer Sprachmerkmale in FORTRAN IV

W. Abel

Kernforschungszentrum Karlsruhe GmbH., Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
ISSN 0303-4003

Abstract

PL/I-FORMAC Program (Precompiler) for FORTRAN IV Language Extensions

FORTRAN IV-compilers for computer installations of different manufacturers contain a so-called PARAMETER statement, which allows to vary array dimensions very easily. This useful statement does not exist in IBM FORTRAN IV so far. Furthermore, with respect to modern software development techniques there is a continuing interest to revise and further improve the programming language FORTRAN IV. Therefore, a precompiler has been written to provide some language extensions in FORTRAN programming. The language extensions are implemented by a pre-processor written in PL/I-FORMAC. The output of this precompiler is always FORTRAN IV.

Kurzfassung

FORTRAN IV-Übersetzer für Rechenanlagen verschiedener Hersteller enthalten eine sogenannte PARAMETER-Anweisung, mit deren Hilfe die Indexgrenzen in Feldvereinbarungen leicht geändert werden können. Eine solche nützliche Anweisung ist in IBM FORTRAN IV bisher nicht vorhanden. Darüberhinaus besteht in Hinblick auf eine moderne Programmierpraxis ein fortgesetztes Interesse daran, die Programmiersprache FORTRAN IV zu überarbeiten und weiter zu entwickeln. Aus diesem Grunde wurde ein Vorübersetzer (precompiler) entwickelt, um einige Spracherweiterungen in der FORTRAN-Programmierung anwenden zu können. Die Spracherweiterungen sind durch einen in PL/I-FORMAC geschriebenen Preprozessor implementiert. Die Ausgabe des Vorübersetzers ist ein in FORTRAN IV konvertiertes Programm.

INHALT

1. Einleitung

Gegenwärtig ist man bestrebt, im Bereich der allgemeinen problemorientierten Programmiersprachen eine Vereinheitlichung und Standardisierung durchzuführen, zumindestens was die Mikrostruktur der Sprache anbetrifft. Aufbau und Leistungsfähigkeit einer Programmiersprache hängen stets davon ab, inwieweit Möglichkeiten zur automatischen Übersetzung der Sprache in die interne Maschinensprache von Rechnern existieren.
Neben der Syntax und Semantik einer Programmiersprache stellt die Pragmatik einen wichtigen Aspekt dar. Hierunter fallen Problemkreise, wie z.B. Aufwendigkeit der Compiler, Effektivität der Programme, leichte Erlernbarkeit der Sprache, Anwendungsgebiete.

Eine leichte und effektive Übersetzung einer Sprache unter Einbeziehung von Optimierungsverfahren finden zu müssen, kommt auch in der von John Backus entwickelten und 1955 vorgestellten Programmiersprache FORTRAN deutlich zum Au›druck. Obwohl FORTRAN wegen ihres algorithmischen Charakters eine weite Verbreitung gefunden hat, ergeben sich dennoch Nachteile, wie z.B. Formulierungsbeschränkungen bezüglich der Indexausdrücke, der Typen von Variablen, oder dem relativ einfachen Prozedurkonzept. FORTRAN IV ist aus der Praxis heraus mehr oder weniger verbal definiert. In Hinblick auf eine moderne Programmierpraxis ist man daran interessiert, Elemente der sogenannten "Strukturierten Programmierung", wie man sie von ALGOL oder PL/I kennt, auch in FORTRAN einzuführen. Dies hat durch Überarbeitung und Weiterentwicklung von FORTRAN IV zu der jetzt vorgeschlagenen Norm FORTRAN 77 geführt/1/.

Der im folgenden beschriebene Precompiler gibt die Möglichkeit, neben der auch in FORTRAN 77 vorhandenen PARAMETER-Anweisung und Block-IF-Anweisung einige darüber hinaus gehende Programmiermöglichkeiten in IBM FORTRAN IV anzuwenden:

- GLOBAL-Anweisung
- Elemente der Strukturierten Programmierung
- Alphanumerische Sprungmarken
- Einfügen von Text mit der SELECT-Anweisung
- Namensgesteuerte Ausgabe von einfachen Variablen
- Generierung spezieller Funktionsausdrücke(formelmäßig)
- STOP-Text und in-line Kommentar

Das Ziel der ersten Version/7/ dieses Programms bestand darin, Anwenderprogramme, die die PARAMETER-Anweisung, die SELECT-Anweisung oder den STOP-Text enthalten, mit IBM FORTRAN IV-Compilern bearbeiten zu können.

Bei der hier vorgestellten Erweiterung bleibt FORTRAN IV als Zielsprache in vollem Umfang gültig, so daß in bestehenden Programmen die zusätzlichen Möglichkeiten leicht genutzt werden können. Im Anhang 2 ist ein einfaches Beispiel angegeben.

Das Programm ist in PL/I-FORMAC/2/ geschrieben, um so vor allem die Auswertung arithmetischer Ausdrücke in der PARAMETER-Anweisung in direkter Weise durchführen zu können. Allerdings muß durch die Wahl dieser Programmiersprache eine Einschränkung bezüglich der Portabilität des Programms in Kauf genommen werden.

2. Bemerkungen zur Arbeitsweise des Precompilers

2.1 Lexikalische/Syntaktische Analyse

Aufgabe des "Precompilers" ist es, die zusätzlichen Sprachmerkmale im vorgegebenen Quellprogramm zu erkennen und gemäß der Übersetzungsvorschrift in IBM-FORTRAN IV-Text/3/ zu übertragen. Syntax und Semantik der beschriebenen Spracherweiterungen legen dabei fest, in welcher Weise einzelne Sprachelemente aus anderen Sprachelementen zusammengesetzt sind bzw. welche Bedeutung die Sprachelemente haben. Es werden die Bedeutungsunterschiede aufgezeigt, die sich aus dem Kontext der Sprachelemente ergeben. Die Syntaxnotation der neuen Sprachelemente basiert dabei auf der als Backus-Naur-Form(BNF)/4/ bekannten Metasprache. Mit Hilfe der unten definierten nichtterminalen Symbole kann so eine kompakte Darstellung der Eigenschaften der neuen Sprachelemente angegeben werden.

Während der syntaktischen Analyse wird das Programm nach nichtterminalen Symbolen oder Schlüsselworten durchsucht; jedem Nichtterminal oder einer Gruppe von Nichtterminalen wird eine Routine zugeordnet, die untersucht, ob eine Ableitung eines Symbols vorliegt. In den Prozeduren wird dann entsprechend der rechten Seiten der zugehörigen Backus-Regel vorgegangen. Nichtterminale Symbole sind als Hilfsmittel zur Sprachbeschreibung aufzufassen und dürfen in diesem Zusammenhang im Zielprogramm nicht auftauchen. Sprachkonstruktionen, die durch Wortsymbole getrennt sind, wie z.B. IF-THEN-ELSE, lassen sich recht einfach mit Hilfe der Methode des rekursiven Abstiegs/5/ analysieren, der in natürlicher Weise eine "parsing machine" (PM) nach Knuth/6/ zugeordnet werden kann.

Die Analyse erfolgt zeilenweise von links nach rechts. Fortsetzungen werden vorher aufgelöst. Das hier verwendete Produktionssystem ist in einer stark vereinfachten Form formuliert worden. Insbesondere wird die innere Struktur der Boolschen oder arithmetischen Ausdrücke nicht weiter untersucht, da es sich um Ausdrücke handelt, die den üblichen FORTRAN-Regeln gemäß niedergeschrieben werden müssen und demnach ohnehin in dem sich anschließenden normalen Übersetzungslauf bearbeitet werden. In manchen Fällen ist eine Erkennung einer Spracherweiterung unmittelbar durch ihr Satzsymbol (bzw. Schlüsselwort) eindeutig möglich und kann in direkter Art und Weise in Abhängigkeit von der Übersetzungsvorschrift in FORTRAN IV-Text übertragen werden.

Als Konsequenz einer solchen "groben" Definition der Spracherweiterungen ergibt sich, daß Linksrekursivitäten nicht auftreten, die sonst zu Endlosschleifen im PM-Programm führen würden. Ferner können Sackgassen während des Analysevorgangs dadurch verhindert werden, daß die betreffenden Algorithmen, von einer eindeutigen Fehlersituation einmal abgesehen, dennoch terminieren, indem (mehr oder weniger willkürlich) angenommen wird, daß eine normale FORTRAN-Anweisung vorliegt. Entspricht in allen anderen Fällen ein Satz der Quelltextkette eindeutig nicht einer der im folgenden beschriebenen Spracherweiterungen, wird davon ausgegangen, daß es sich um einen gültigen FORTRAN-Satz handelt, der somit unverändert auf das Ausgabefile übertragen wird. Insbesondere werden Kommentare und FORMAT-Anweisungen unmittelbar übernommen. Allerdings hat ein solches direktes Vorgehen zur Folge, daß der linke und/oder rechte (beschränkte) Kontext einer gerade untersuchten Teil-

kette in Betracht gezogen werden muß.

Da diesem Programm in Strenge kein automatentheoretisches Modell zugrundeliegt, wird in manchen Fällen das Leerzeichen zur Trennung der Morpheme als syntaktisches Zeichen benutzt, während es sonst eine rein typographische Funktion hat, um nicht zu sehr von der gewohnten FORTRAN-Schreibweise abweichen zu müssen. Aus diesem Grunde kann auch eine Formatfreiheit der Quelle nicht vollständig eingeführt werden. Lediglich der weiter unten (siehe 3.6) beschriebene in-line-Kommentar bildet dabei eine Ausnahme.

2.2 Eingabe/Ausgabe

Der erste Schritt der Analyse besteht darin, das Quellprogramm auf eine temporäre Datei zu übertragen. Dabei wird gegebenenfalls die SELECT-Bibliotheksfunktion(siehe 3.5.2) ausgeführt, so daß dann der vollständige Programmtext der Analyse unterworfen werden kann.

Typographische Merkmale der Niederschrift des Quellprogramms, wie Zwischenräume oder Einrückungen gehen teilweise verloren: Das in FORTRAN IV übertragene Programm wird in der allgemein üblichen spaltengerechten Form ausgegeben. Bearbeitete Anweisungen mit Fortsetzung werden zeilenfüllend angeordnet.

Das FORTRAN IV-Programm wird auf eine temporäre sequentielle Datei geschrieben. Eine Liste des Quellprogramms kann über das Ausgabefile LISTE (siehe 4.) angefordert werden.

2.3 Fehlerbehandlung

Syntaktische Fehler bezüglich der hinzugefügten Sprachelemente werden angezeigt bzw. in einigen Fällen durch "vernünftige" Annahmen korrigiert. Nicht gefundene Fehler sollen dann in der sich anschließenden Weiterbearbeitung mit einem FORTRAN-Compiler erkannt werden.

2.4 Namenskonventionen

Wenn nicht ausdrücklich erklärt, gelten für das Schreiben von Ausdrücken, das Bilden von Variablennamen oder die Formulierung von Kontrollstrukturen, wie schon oben erwähnt, die allgemein bekannten FORTRAN-Regeln. Besonderheiten der syntaktischen bzw. lexikalischen Analyse werden im folgenden bei der Beschreibung der einzelnen Sprachmerkmale erläutert.

3. Beschreibung der FORTRAN IV Spracherweiterungen

Die im folgenden beschriebenen Spracherweiterungen werden im Sinne einer einfachen top-down-syntaxorientierten Analyse mit Hilfe der Backus-Naur-Form definiert.
Ein Begriff, der auf der linken Seite des Produktionszeichens ::= steht, wird durch den rechts stehenden Ausdruck beschrieben. Ein sogenanntes Metasymbol ist eine frei wählbare Bezeichnung und wird in spitze Klammern <..> eingeschlossen.
Da in einigen Produktionen manche Teile wahlweise vorkommen können, gelten hierfür folgende Vereinbarungen:

$\{..\}_n^m$ Allgemein: n- bis m-fache Wiederholung ($n \leq m$) eines Terms

[..] Ein in eckige Klammern eingeschlossener Term darf fehlen.

$\{..\}^*$ Dieser Teil tritt nicht oder mehrmals auf.

Alternative Terme sind durch | voneinander getrennt. Wegen der Anbindung an FORTRAN (keine Formatfreiheit der Quelle) dient der Pfeil ↓ dazu, einen Zeilenvorschub anzudeuten. Ein in einer Definition zwingend vorgeschriebenes Leerzeichen ist durch das Unterstreichungszeichen _ gekennzeichnet.

Insbesondere bedeuten zur kürzeren Darstellung:

<A> Arithmetischer Ausdruck
<B> Boolscher Ausdruck (Primärausdruck, Vergleichsausdruck)
<M> Sprungmarke (alphanumerische Marke(siehe 3.3) oder FORTRAN-Label)
<S> Anweisung oder eine Folge von Anweisungen (bedingt oder unbedingt)

Diese nichtterminalen Symbole führen immer auf einen entsprechenden FORTRAN-Ausdruck, so daß sie nicht weiter erklärt werden müssen. In <S> können natürlich auch Spracherweiterungen geschrieben werden.

3.1 Variable Dimensionierung

3.1.1 PARAMETER-Anweisung

Programme, die z.B. in Honeywell FORTRAN geschrieben sind, können die sogenannte PARAMETER-Anweisung enthalten, mit deren Hilfe sehr leicht die Dimensionierungen von Feldern geändert werden können. Solche Programme ohne Änderung auf IBM-FORTRAN IV übertragen zu können, war die Aufgabe der ursprünglichen Version/7/ dieses Programms. So bestand die Möglichkeit, allen IBM-FORTRAN IV-Programmen durch Hinzufügen dieser Anweisung diese variable Dimensionierung zugänglich zu machen.

<PARAMETER-Anweisung> ::= PARAMETER <Wertzuweisungsliste>

<Wertzuweisungsliste> ::= {<Wertzuweisung>,}* <Wertzuweisung>
<Wertzuweisung> ::= <Variable>=<A>

Die Syntax entspricht bis auf eine Klammerung der Wertzuweisungsliste der in FORTRAN 77 gegebenen Definition. Es können eine oder mehrere Wertzuweisungen vorkommen. Bei einer Liste sind die einzelnen Wertzu-

weisungen durch Komma voneinander zu trennen. Aus der Anwendung heraus sind nur FORTRAN INTEGER Variablen erlaubt. Leerzeichen innerhalb der Wertzuweisungen werden ignoriert. PARAMETER-Karten können nicht fortgesetzt werden.
Hervorzuheben ist, daß in der Wertzuweisung einfache arithmetische Ausdrücke zugelassen sind, deren Auswertung durch den Übergang von PL/I nach FORMAC in einfacher und direkter Weise durchgeführt werden kann, ohne diese Wertzuweisungen im üblichen Sinne übersetzen und ausführen zu müssen. Das folgende Beispiel zeigt die Anwendung der PARAMETER-Anweisung.

```
      PARAMETER  N = 2
      PARAMETER  IRR=N*(N+1)/2
      PARAMETER  NK = 26, NE=NK*N
C
      DIMENSION A(N,5),B(IRR),C(IRR,N),D(NK,N)
      DIMENSION E(NK),F(NE,NE,NE),H(3,2,NK)
```

Die Ersetzung der Indexvariablen wird in allen Vereinbarungsanweisungen, die mit dem Schlüsselwort COMMON, REAL, INTEGER, COMPLEX, LOGICAL und DIMENSION beginnen, durchgeführt.
Hierbei ist hervorzuheben, daß nur innerhalb von Indexausdrücken, d.h. innerhalb von "Dimensions"-Klammern, eine Ersetzung durchgeführt wird, ganz im Gegenteil zu anderen Übersetzern, wie auch FORTRAN 77, wo jedes Auftreten der betreffenden Variablen erkannt und durch ihren numerischen Wert substituiert wird. Insbesondere können solche Variablen deshalb auch in COMMON-Anweisungen stehen. Der Operandenteil der PARAMETER-Anweisung wird in gültige FORTRAN-Anweisungen umgewandelt, so daß diese Problemparameter dem Programm zur Ausführungszeit zur Verfügung stehen, z.B. in Unterprogrammaufrufen, als DO-Begrenzer, usw.
Um eine Übersetzung des FORTRAN-Programms mit dem IBM-H-Compiler zu ermöglichen, werden diese Anweisungen vor der ersten ausführbaren Anweisung eingeordnet. Sogenannte Statement-Funktionen werden allerdings nicht erkannt. Die obigen Anweisungen haben nach der Ersetzung die folgende Form:

```
      DIMENSION A(2,5),B(3),C(3,2),D(26,2)
      DIMENSION E(26),F(52,52,52),H(3,2,26)
C
      N = 2
      IRR=N*(N+1)/2
      NK = 26
      NE=NK*N
```

3.1.2 GLOBAL-Anweisung

Während bei der Anwendung der PARAMETER-Anweisung eine Ersetzung der Feldgrenzen nur in der betreffenden Programmeinheit, d.h. Hauptprogramm oder Unterprogramm, durchgeführt wird, bietet die GLOBAL-Anweisung dem Benutzer die Möglichkeit, Indexgrenzen global (sinnvoller Weise im Hauptprogramm) zu definieren. Dann sind bei nachträglicher Problemanpassung nur an einer Stelle die jeweiligen Wertzuweisungen zu korrigieren. Die beiden Anweisungstypen dürfen gemischt auftreten. Die Reihenfolge der Anweisungen untereinander muß dabei eine korrekte

sequentielle Abarbeitung ermöglichen, weil sonst die Variablen nicht definiert sind. Beim Übergang von einer Programmeinheit zu einer anderen werden alle GLOBAL-Anweisungen in einer Liste (Stapel) mitgeführt, nachdem diese so sortiert wurde, daß alle GLOBAL-Anweisungen vor den PARAMETER-Anweisungen auftreten. In der neuen Programmeinheit geltende PARAMETER-Anweisungen werden dann an einen so entstehenden Listenkopf angefügt.
Die Syntax der GLOBAL-Anweisung entspricht derjenigen der PARAMETER-Anweisung.

```
      GLOBAL M = 3
      GLOBAL L=15, NN=M*(L+1)
C
      DIMENSION A(M,5),B(L),C(NN,M)
```

Neben der Programmierung der GLOBAL-Anweisung im Quelltext können globale Indexvariable ganz oder teilweise auch über das Datenfile GETDIM definiert werden, indem jedoch nur die Wertzuweisungsliste der GLOBAL-Anweisung als Zeichenkette eingelesen wird. Fehlt eine entsprechende DD-Karte, so wird vom Preprozessor eine solche Eingabe nicht erwartet. Gemäß dem obigen Beispiel müßten dann die folgenden Karten anstelle der GLOBAL-Anweisungen dem Kontrollkartensatz hinzugefügt werden:

```
//G.GETDIM DD *
M=3, L=15, NN=M*(L+1)
```

3.2 Strukturiertes Programmieren in FORTRAN

Neben der Anwendung der top-down-Methode und dem Bemühen, einem Programm eine möglichst modulare Struktur zu geben (bottom-up-Entwurf), liegt das Ziel der "Strukturierten Programmierung" darin, Techniken der Programmierung bereitzustellen, um zu "guten" Programmen zu gelangen. Einer der Problemkreise der Strukturierten Programmierung hat die Vermeidung von GOTO-Anweisungen/8/ zum Inhalt, wodurch eine Beziehung zwischen dem statischen und dynamischen Verhalten eines Programms aufgezeigt werden kann. Dies führt mit Unterstützung der oben genannten Entwurfsstrategien zu einer größeren Klarheit und Verständlichkeit eines Programms, d.h. zu einer höheren Wartungsfreundlichkeit. Wichtige Stufen sind hierbei die Erlernbarkeit, Flexibilität und Erweiterbarkeit eines Algorithmus. Daher sind auch in FORTRAN Änderungen wünschenswert zur Unterstützung der Strukturierten Programmierung. Neben der IF-THEN-ELSE Kontrollstruktur ist eine DO-WHILE-Iteration sinnvoll. Eine Programmiersprache sollte deshalb aus dem gleichen Grund möglichst wenig Strukturelemente besitzen. Böhm und Jacopini/9/ haben gezeigt, daß 3 elementare Strukturelemente zur Programmierung eines Algorithmus ausreichen: Anweisungsblock, Alternative und Iteration. Eine "GOTO-Freiheit" ist allerdings nicht leicht zu erzeugen, jedoch wird auch sparsames Umgehen mit GOTO-Anweisungen die Lesbarkeit (Transparenz) eines Programms vergrößern. Da auch rückwärts gesprungen werden kann, ist der Nachweis der dynamischen Endlichkeit eines Algorithmus außerordentlich schwer zu erbringen. In den meisten Fällen läßt sich jedoch ein solcher Sprung durch Einführung einer DO-WHILE-Konstruktion vermeiden.

3.2.1 Wiederholungsanweisungen

3.2.1.1 Bedingungsschleifen

Die von anderen Programmiersprachen wie PL/I her bekannte Wiederholungsanweisung gibt die Möglichkeit eine Iteration auszuführen, d.h. eine Folge von Anweisungen wird so lange ausgeführt, bis eine bestimmte Bedingung einen Abbruch der Wiederholung bewirkt. Im folgenden werden zwei verschiedene Typen der Bedingungsschleifen beschrieben. Die Syntax ist für beide die gleiche, sie unterscheiden sich nur durch die Wortsymbole WHILE und UNTIL. Semantisch ergibt sich der Unterschied aus dem Schleifenendetest. In beiden Fällen ist eine Nullschleife möglich, d.h. die Ausführung kann durch geeignetes Setzen der Anfangsbedingung von vornherein unterbunden werden.
In den Anweisungen des Schleifenrumpfes muß der Wert mindestens einer Variablen der Schleifenbedingung verändert werden. Die Veränderung der Schleifenvariablen wird man im allgemeinen am Ende des Schleifenrumpfes durchführen.

<DO WHILE-Anweisung>::= [<M>]_DO_WHILE(<B>)↓<S>↓ENDDO

<DO UNTIL-Anweisung>::= [<M>]_DO_UNTIL(<B>)↓<S>↓ENDDO

Das folgende Beispiel zeigt eine Anwendung einer solchen Anweisung:

```
DO WHILE(X.LT.0.)
  ...
ENDDO
```

In Hinblick auf eine effektive Übersetzung soll noch erwähnt werden, daß in beiden Fällen eine optimale Schleifenstruktur ausgewählt wurde/10/.

Die hier beschriebenen Wiederholungsanweisungen haben den gleichen Abschluß. Die Sprungmarken werden in einer internen Liste gespeichert und je nach Auftreten der Abschlußanweisung wird die zugehörige Absprungmarke nach dem LIFO-Prinzip(last-in-first-out) zur korrekten Bearbeitung der Schachtelungstiefe ausgewählt. Ist am Ende der Verarbeitung einer Programmeinheit, d.h. nach der END-Anweisung, die Schachtelungstiefe nicht null, so wird eine diesbezügliche Fehlernachricht ausgegeben.

3.2.1.2 FOR-Laufanweisung

Während in FORTRAN IV nur reine Zählschleifen möglich sind, bietet diese Wiederholungsanweisung den Vorteil, daß als Laufvariable auch Variable vom Typ REAL verwendet werden können. Die Syntax dieser Anweisung kann folgendermaßen beschrieben werden:

<FOR-Anweisung> ::= [<M>]_<FOR-Klausel>↓<S>↓ENDFOR

<FOR-Klausel> ::= FOR <Laufvariable>=<Lauflistenelement>
<Lauflistenelement> ::= <A>_TO_<A>[_BY_<A>][_WHILE(<B>)]

Der zu wiederholende Anweisungsteil <S> wird so lange abgearbeitet, bis der Zähler <Laufvariable> den auf TO folgenden Endwert überschritten hat. Die Zählung wird mit der durch BY <A> gegebenen Schrittweite ausgeführt. Eine Wiederholungsanweisung mit Bedingung hat dann z.B. die folgende Form:

```
FOR X=0. TO 2*Y BY 2.5 WHILE(Z.GT.0.)
  ...
ENDFOR
```

Im Zusammenhang mit der Anschlußbehandlung muß die Reihenfolge der Auswertung der Terminationsbedingungen beachtet werden. Vor jedem Durchlauf wird zuerst geprüft, ob die Laufvariable schon den Endwert angenommen hat. Danach wird die WHILE-Bedingung abgefragt. Nach dem gesamten Durchlauf ist die Laufvariable je nach Inkrementwert größer oder kleiner als der Endwert, unabhängig von dem WHILE-Element. Eine Schleife wird demnach überhaupt nicht ausgeführt, wenn die Laufvariable von vornherein größer als der Endwert ist. Eine Ausführung kann aber auch durch das Hinzufügen des WHILE-Elements so gesteuert werden, daß ein Abbruch der Iteration bewirkt wird oder eine Nullschleife vorliegt.
In Hinblick auf eine eindeutige Erkennung müssen die Wortsymbole TO und BY jeweils durch ein Leerzeichen eingeschlossen sein.
Folgt nach FOR bis zum nächsten Schlüsselwort keine Wertzuweisung, so wird eine entsprechende Fehlernachricht ausgegeben. Die Reihenfolge des TO-Elements und des BY-Elements ist gleichgültig. Fehlt das BY-Element, so wird BY 1 ergänzt. Bei der Schrittweite 0 ergibt sich eine Endlosschleife. Fehlt dagegen das TO-Element, so sollte die WHILE-Bedingung vorhanden sein oder ein anderer Durchlauftest benutzt werden, da sonst die Schleife nicht terminiert. Insbesondere ist eine

negative Schrittweite erlaubt, so daß in einfacher Weise die Schleifen rückwärts durchlaufen werden können. Schachtelungen sind möglich. Nach dem Wiederholungsteil <S> muß die Abschlußanweisung ENDFOR geschrieben werden.

3.2.1.3 NEXT, EXIT

Da bei Verwendung der hier beschriebenen Wiederholungsanweisungen keine Sprungmarken benötigt werden, muß eine Möglichkeit gegeben sein, die DO-Schleife zu beenden bzw. zur nächsten Iteration überzugehen, ohne daß der gesamte Schleifenkörper durchlaufen wird. Die hierfür dienenden Anweisungen EXIT bzw. NEXT/11/ können überall dort auftreten, wo in FORTRAN IV eine GOTO-Anweisung stehen darf.

Um auch bei Schachtelungen einen gesteuerten Ablauf zu ermöglichen, d.h. bei "labeled loops", kann die Marke der betreffenden Schleife nach dem ausgewählten Schlüsselwort niedergeschrieben werden:

NEXT [<M>] EXIT [<M>]

3.2.2 Bedingte Programmverzweigung (Block-IF)

Zusätzlich zu den in FORTRAN IV gegebenen Kontrollstrukturen steht dem Benutzer insbesondere zu einem möglichst GOTO-freien Programmieren eine neue Kontrollstruktur zur Verfügung. In Übereinstimmumg mit FORTRAN 77 hat die sogenannte Block-IF-Anweisung die folgende Form:

```
<IF-Anweisung>::= [<M>]_<IF-Klausel>[↓<ELSE-Klausel>]↓<Abschluß>

<IF-Klausel>    ::= IF(<B>) THEN
<ELSE-Klausel>  ::= <S>↓ELSE
<Abschluß>      ::= <S>↓ENDIF
```

Mögliche Formen der bedingten Anweisung sind dann:

zweiseitige Auswahl:

```
IF(X.EQ.0.) THEN
 ..
 ..
ELSE
 ..
 ..
ENDIF
```

einseitige Auswahl:

```
IF(X.EQ.0.) THEN
 ..
 ..
ENDIF
```

Die obige Form der IF-Anweisung ist nur zweiwertig. Da durch Schachtelung auch eine Fallunterscheidung gemacht werden kann, wurde die verkürzte Schreibweise mit Hilfe der ELSEIF-Klausel eingeführt. Diese multiple Bedingung entspricht der CASE-Formel in ALGOL. Die ELSEIF-Klausel tritt dann entsprechend oft auf. In jedem Fall muß hier die ELSE-Klausel als Ausweichformel angegeben sein.

```
<IF-Anweisung>::= [<M>]_<IF-Klausel>
                  [{↓<ELSEIF-Klausel>}*↓<ELSE-Klausel>]↓<Abschluß>

<ELSEIF-Klausel> ::= <S>↓ELSEIF(<B>) THEN
```

Eine Fallunterscheidung kann somit das folgende Aussehen haben:

```
IF(X.EQ.0.) THEN
 --
ELSEIF(X.GT.A) THEN
 --
ELSEIF(X.GT.B) THEN
 --
ELSEIF(X.GT.C) THEN
 --
ELSE
 ..
 ..
ENDIF
```

Die IF-Anweisungen können auch hier bis zu einer Tiefe von 99 geschachtelt werden. Von den dann im vorübersetzten FORTRAN-Programm auftretenden numerischen Labeln (siehe 3.3) kann abgelesen werden, um die wievielte IF-Abfrage bei gegebener Schachtelungstiefe es sich handelt.

3.3 Alphanumerische Marken

Eine weitere Besonderheit des Precompilers liegt darin, daß alphanumerische Sprungziele programmiert werden können. Die maximale Länge ist dabei willkürlich auf 20 Zeichen festgelegt. Solche Marken sind dann sinnvoll, wenn das Sprungziel auf einen bestimmten Algorithmus oder Verarbeitungsteil des Programms hindeuten soll. Den im Laufe der Analyse erkannten alphanumerischen Marken wird ein numerischer Wert zugeordnet und in einer internen Tabelle mitgeführt.

<Marke>::= :<Bezeichnungsausdruck>:

z.B.

```
IF(N.GT.0) GOTO :EINLESEN:
  ...
  ...
:EINLESEN: READ(5,*) LISTE
```

Marken treten überall dort auf, wo sie auch von FORTRAN IV her erlaubt sind. Wegen der Länge der Marke kann die Ziel-Anweisung auch leer sein. In diesem Fall wird eine CONTINUE-Anweisung generiert.

3.4 Namensgesteuerte Ausgabe von INTEGER und REAL Zahlen

Eine weitere Hilfe beim Programmtest kann die sogenannte namensgesteuerte Ausgabe von einfachen Variablen sein, die den Benutzer vom Schreiben von entsprechend aufwendigen WRITE-Anweisungen entlastet. Darunter versteht man eine Ausgabeform, bei der Name und Wert einer Variablen in Form einer "Wertzuweisung" protokolliert werden. Da der Datentyp nicht abgeprüft wird, stehen zur Ausgabe von INTEGER bzw. REAL Variablen oder einfachen Feldelementen verschiedene Anweisungen zur Verfügung:

<Ausgabe-Anweisung> ::= OUTINT<Variablenliste>|OUTREAL<Variablenliste>

<Variablenliste> ::= {<Variable>,}*<Variable>
<Variable> ::= <einfache Variable>|<indizierte Variable>

Folgende Ausgabe-Anweisungen sind dann z.B. möglich:

```
OUTINT I1,I2,I3,I4,I5,L,M,K(10),J

OUTREAL X,Y,Z,W(11),U,V
```

Diese Anweisungen werden in die folgenden WRITE-Anweisungen übersetzt; die FORMAT-Label werden durchgezählt:

```
      WRITE(6,90000) I1,I2,I3,I4,I5,L,M,K(10),J
90000 FORMAT('0',
     *'       I1 =',I10,'       I2 =',I10,'       I3 =',I10,
     *'       I4 =',I10,'       I5 =',I10,'        L =',I10/,
     *'        M =',I10,'    K(10) =',I10,'        J =',I10)
```

```
      WRITE(6,90001) X,Y,Z,W(11),U,V
90001 FORMAT('0',
     *'        X =',E12.5,'        Y =',E12.5,'        Z =',E12.5,
     *'    W(11) =',E12.5,'        U =',E12.5,'        V =',E12.5)
```

3.5 Makro-Anweisungen

3.5.1 Generierung spezieller Funktionen

In manchen Fällen erscheint es nützlich, die Möglichkeiten der FORMAC-Sprache zur symbolischen Bearbeitung von mathematischen Ausdrücken auszunutzen. Die Generierung spezieller Funktionen, wie Legendre-, Hermitepolynomen oder Taylorreihen ist gerade dann von Vorteil, wenn die einschlägigen Tafelwerke nur eine Auswahl bzgl. der Ordnung der Entwicklung und der Argumente enthalten und außerdem die formelmäßige Struktur der Polynome interessiert. Da FORMAC einen Differential-Operator enthält und viele Funktionen überdies rekursiv definiert werden können, lassen sich solche Entwicklungen für beliebiges Argument oder Ordnung leicht bilden.

$GENERATE <Funktionsname> [<Parameterliste>]

Für jeden Funktionsnamen muß allerdings eine Routine gleichen Namens im Analyseprogramm bisher fest eingebaut sein. Einige Beispiele seien hier nur zur Demonstration dieser Anweisung angeführt:

$GENERATE LEGENDRE <Grad des Polynoms>

$GENERATE HERMITE <Grad des Polynoms>

Diese Aufrufe generieren jeweils einen Funktionsausdruck f(x) mit der unabhängigen Variablen x. Mit Hilfe des Aufrufs

$GENERATE GRADIENT <Funktionsausdruck> <Diff.-Vorschrift>

<Diff.-Vorschrift>::= {<Variable>,<Ordnung>,}$_0^{m-1}$<Variable>,<Ordnung>

kann der gemischte Differentialquotient nach m Variablen eines Funktionsausdrucks bestimmt werden, z.B.:

$GENERATE GRADIENT SIN(X)*Y**5 X,3,Y,2

Der Ausdruck soll nach der Variablen x dreimal und nach y zweimal abgeleitet werden. Der aus dem FORMAC-Ausdruck der Funktion gewonnene PL/I-Ausdruck wird als Wertzuweisung in den Programmtext eingefügt, wobei als linke Seite der Wertzuweisung ein beschreibender Name in Abhängigkeit von der aufgerufenen Funktion eingesetzt wird. Im Falle des obigen Beispiels wird als Ergebnis die folgende FORTRAN-Anweisung anstelle des GENERATE-Aufrufs in das Programm eingefügt:

```
GD01= -Y**3*COS(X)*20
```

3.5.2 SELECT-Bibliotheksfunktion

Wenn während der Analyse eine SELECT-Anweisung identifiziert wird, kann externer Text in den Quelltext eingefügt werden. Die SELECT-Anweisung ist folgendermaßen zu schreiben:

$SELECT <member>

Das <member> kann irgendwelchen Text, z.B. Unterprogramme, Anweisungsfolgen, COMMON-Blöcke oder auch Preprozessor-Anweisungen enthalten. Der externe Text muß ein Member der unter dem File-Namen MAKRO referierten Benutzerbibliothek sein. Die Übertragung der einzelnen Member wird von einer in IBM/370 ASSEMBLER geschriebenen Routine übernommen; die Textzeilen werden jeweils in die sequentielle temporäre Datei MCARD kopiert und vom Übersetzerprogramm dann an der betreffenden Stelle in den Quelltext eingefügt.

3.6 STOP-Text und in-line-Kommentar

Neben der in FORTRAN IV üblichen Kommentarkarte kann hier auch beliebiger Text in eine Programmzeile zur besseren Dokumentation des Programms geschrieben werden. Als Trennzeichen ist hierfür das Semikolon vereinbart. Vor der Übertragung bzw. Analyse der betreffenden Zeile wird der Text gelöscht. Tritt das Semikolon in Spalte 1 auf, so wird eine solche Zeile wie eine gewöhnliche Kommentarkarte behandelt. Da eine Zeile immer nur eine Anweisung enthalten kann, ist ein Kennzeichnen des Textendes nicht notwendig, zumal Kommentare "innerhalb" der Anweisungen nicht auftreten sollen.

Als eine einfache DEBUG-Möglichkeit ist der sogenannte STOP-Text hervorzuheben, d.h. Text, der nach einer STOP-Anweisung steht, wird bei der Ausführung dieser Anweisung über die Druckausgabe des Programms protokolliert. Hierzu wird ein CALL-Aufruf generiert, das zugehörige Unterprogramm wird der "MAKRO"-Bibiliothek entnommen und nach der ersten END-Anweisung (im Anschluß an das Hauptprogramm) in den Programmtext eingefügt. Die Anweisung

IF(X.EQ.0.) STOP ;<Fehlernachricht>

wird übersetzt in

IF(X.EQ.0.) CALL STOP('<Fehlernachricht>',<n>)

Für <n> wird die Länge des Textes in Bytes eingesetzt.

3.7 Generierung von FORTRAN-Marken

Da die hier beschriebenen Wiederholungsanweisungen und die Block-IF-Anweisung in FORTRAN IV-Anweisungen übersetzt werden, ergibt sich das Problem der Bildung von FORTRAN-Marken. Diese werden automatisch nach der folgenden Vorschrift generiert:

<FORTRAN-Marke> = <Basiszahl + Blocknummer + Schachtelungstiefe>

Blocknummer = Blocknummer + 100 * Schachtelungstiefe

Damit kann aus der erzeugten Marke folgendes abgelesen werden: (a) die erste Stelle ist charakteristisch für den Anweisungstyp, (b) die nächsten beiden Stellen zählen die Anzahl der "Blöcke" auf gleichem Schachtelungniveau, (c) die letzten beiden Ziffern geben direkt die Schachtelungstiefe wieder. Bei dieser Art der Zählung sind Blockzahl und Schachtelungstiefe auf 99 beschränkt; im minimalen Fall bedeutet dies, das nur 99 Anweisungen eines Typs geschrieben werden können. Wird auf diese spezielle Art der Anweisungs- bzw. Blockzählung verzichtet und die Schachtelungstiefe auf 9 reduziert, so erhöht sich die Zahl der Anweisungen pro Programmeinheit auf 999.
Die Basiszahlen sind für die hier beschriebenen Programmstrukturen folgendermaßen festgelegt:

(1) Für DO WHILE und DO UNTIL : 10000, 20000
(2) Für IF-THEN-ELSE : 30000, 40000
(3) Alphanumerische Marken : 60000
(4) Für FOR-TO-BY-WHILE : 10000, 20000, 50000
(5) FORMAT-Marke für OUTINT oder OUTREAL : 90000

Die Basiszahl wird deshalb mit maximaler Stelligkeit gewählt, um mit vorhandenen FORTRAN-Marken möglichst nicht in Konflikt zu geraten; bei der Programmerstellung ist deshalb darauf zu achten, daß nur Marken <10000 vorkommen.

Da Preprozessor-Anweisungen auch mit einer Marke beginnen können, wird im Falle der Bedingungsanweisungen an entsprechender Stelle eine CONTINUE-Anweisung eingefügt.

4. Aufruf des Precompilers

Die zum Aufruf des Precompilers notwendigen Kontrollkarten sind in der katalogisierten Prozedur PREFOR (siehe Angang 1) zusammengefaßt.

```
// EXEC PREFOR[,MACLIB=name][,C=n][,V=v]
//P.SYSIN DD *
      .
      FORTRAN IV Quellprogramm + Preprozessor-Anweisungen
      .
```

Wird in den Kontrollkartensatz die DD-Karte

```
//P.LISTE DD SYSOUT=*
```

eingefügt, so erhält man eine Liste der Precompiler-Eingabe. Mit Hilfe der Steuerkarten

```
//*FORMAT PU,DDNAME=PUNCH,FORMS=STANZ
//P.PUNCH DD SYSOUT=B,DCB=BLKSIZE=80
```

wird eine Kartenausgabe des vorübersetzten Programms erzeugt. Das in Standard FORTRAN übertragene Programm kann der Benutzer über die Zwischendatei &&PRE zur weiteren Verarbeitung ansprechen.

z.B.:

```
// EXEC FGC ( oder FHC, F7C )
//C.SYSIN DD DSN=&&PRE,DISP=(OLD,DELETE)
```

Die Prozedur enthält einige Schlüsselwort-Parameter, von denen 3 für den Benutzer von Bedeutung sind und die gegebenenfalls überschrieben werden müssen:

1) Benutzerbibliothek
Die Makro-Bibliothek muß über den MACLIB-Parameter angegeben werden, wenn mit Hilfe der SELECT-Anweisung externer Text in das Programm eingefügt werden soll.

2) Länge des Quellprogramms
Über den symbolischen Parameter C ist es möglich, die zur Aufnahme des Quellprogramms benötigte Anzahl n der Zylinder auf der Platte zu überschreiben. Der Standardwert ist auf 2 Zylinder festgesetzt und ist für Programme mit etwa 6000 Karten ausreichend.

3) Programmversion
Ein Aufruf der Prozedur mit V=S (Standardwert) entspricht einer Version v des Programms, bei der lediglich die PARAMETER-Anweisung, die SELECT-Anweisung und der STOP-Text bearbeitet werden. Im anderen Fall ist V=L zu setzen.

Literatur

/1/ ANS X3.9-1978 Programming Language FORTRAN

/2/ R.Tobey et al., PL/I-FORMAC Interpreter User's Manual,
IBM Contributed Program Library
306D-03.3.004, 1967

/3/ IBM SYSTEM /360 and /370,
FORTRAN IV Language,
Order No. GC28-6515-8

/4/ J.W.Backus et al., Num Math. 4(1963), p.420-423

/5/ W.Schneider, Compiler, De Gruyter, Berlin, New York 1975

/6/ D.E.Knuth, Acta Informatica, 1(1971), p.79-110

/7/ W.Abel, KfK 2481 (1977)

/8/ E.W.Dijkstra, Comm. of the ACM, 11(1968), No.3, p.147-148

/9/ C.Böhm,G.Jacopini, Comm. of the ACM, May 1966, p.366-371

/10/ H.Wettstein, Sytemprogrammierung, Hauser, München (1972)

/11/ A.J Cook, L.J.Shustek, MORTRAN2, unveröffentlicht (1975)

Anhang 1: Prozedur PREFOR

```
//PREFOR PROC MACLIB='OBJ.DUMMY',C=2,LIB='LOAD.IAK',V=S
//*******************************************************************************
//*** PRE-PROCESSOR FOR FORTRAN IV LANGUAGE EXTENSIONS              *
//*******************************************************************************
//P EXEC PGM=F77&V
//*** LOAD MODULE (PL/I-FORMAC PROGRAM)
//STEPLIB DD DISP=SHR,DSN=&LIB
//SYSABEND DD DUMMY
//SYSPRINT DD SYSOUT=*,
//   DCB=(LRECL=125,BLKSIZE=3879,RECFM=VBA)
//*** MACRO LIBRARY
//MAKRO DD DISP=SHR,DSN='TSO352.MACFORT.DATA'
//      DD DISP=SHR,DSN=&MACLIB
//MCARD DD DSN=&&DDDD,UNIT=SYSDA,SPACE=(TRK,(2,1)),DISP=(,DELETE)
// DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//*** PREPROCESSED TEXT (STANDARD FORTRAN)
//OUT DD DSN=&&PRE,UNIT=SYSDA,DISP=(NEW,PASS),
// DCB=(LRECL=80,RECFM=FB,BLKSIZE=3120),SPACE=(CYL,(&C,1))
//*** INPUT SOURCE
//IN DD DSN=&&SRCE,UNIT=SYSDA,SPACE=(CYL,(&C,1)),DISP=(,DELETE),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//FORIN DD DDNAME=SYSIN
// PEND
```

Anhang 2: Beispiel

Quellprogramm mit Preprozessor-Anweisungen

```
      PARAMETER N= 3          ;GRAD DES POLYNOMS
      PARAMETER M = N+1
      REAL*4 KOEFF(M)
C
C     BESTIMMUNG EINER REELLEN NULLSTELLE EINES POLYNOMS
C     NACH DEM BISECTIONSVERFAHREN
C
      READ(5,*) KOEFF         ;KOEFFIZIENTEN DES POLYNOMS
      READ(5,*) XL,XR         ;INTERVALLGRENZEN
C
      EPS=0.000001            ;FEHLERSCHRANKE
      ARG=XL
      A=HORNER(ARG,KOEFF)
      DO WHILE(XR-XL.GT.EPS) ;ITERATION
      ARG=(XL+XR)/2.          ;INTERVALLMITTE
      B=HORNER(ARG,KOEFF)
      IF(A*B.LT.0.) THEN
      XR=ARG                  ;NEUE RECHTE GRENZE
      ELSE
      XL=ARG                  ;NEUE LINKE GRENZE
      ENDIF
      ENDDO
      XNULL=(XL+XR)/2.        ;ERMITTELTE NULLSTELLE
      OUTREAL XNULL           ;AUSGABE
      STOP
      END


      REAL FUNCTION HORNER(X,KOEFF)
      PARAMETER N= 3          ;GRAD DES POLYNOMS
      PARAMETER M = N+1
      REAL*4 KOEFF(M)
C
C     BESTIMMUNG DES FUNKTIONSWERTES NACH DEM HORNER-SCHEMA
C
      H=0.
      FOR I= M TO 1 BY -1
      H=H*X+KOEFF(I)
      ENDFOR
      HORNER=H
      RETURN
      END
```

Vorübersetztes Programm

```
      REAL*4 KOEFF(4)
C
C     BESTIMMUNG EINER REELLEN NULLSTELLE EINES POLYNOMS
C     NACH DEM BISECTIONSVERFAHREN
C
C $$ ARRAY LIMITS INSERTED FROM PARAMETER CARD
      N=3
      M=N+1
C $$END
      READ(5,*) KOEFF
      READ(5,*) XL,XR
      EPS=0.0001
      ARG=XL
      A=HORNER(ARG,KOEFF)
      GOTO 20101
10101 CONTINUE
      ARG=(XL+XR)/2.
      B=HORNER(ARG,KOEFF)
      IF(.NOT.(A*B.LT.0.)) GOTO 30101
      XR=ARG
      GOTO 40101
30101 CONTINUE
      XL=ARG
40101 CONTINUE
20101 IF((XR-XL.GT.EPS)) GOTO 10101
      XNULL=(XL+XR)/2.
      WRITE(6,90000) XNULL
90000 FORMAT('0',
     *'    XNULL =',E12.5)
      STOP
      END


      REAL FUNCTION HORNER(A,KOEFF)
      REAL*4 KOEFF(4)
C
C     BESTIMMUNG DES FUNKTIONSWERTES NACH DEM HORNER-SCHEMA
C
C $$ ARRAY LIMITS INSERTED FROM PARAMETER CARD
      N=3
      M=N+1
C $$END
      H=0.
      I = M
      GOTO 50201
10201 I = I+(-1)
50201 IF(-1*(I-1).GT.0) GOTO 20201
      H=H*A+KOEFF(I)
      GOTO 10201
20201 CONTINUE
      HORNER=H
      RETURN
      END
```