Literatur

[1] *Maier, H.:* Datenfernverarbeitung – Entwicklung, Status, Zukunftsaussicht. Online (1975), S. 564, 680, 766.

[2] *Hangele, I.:* Auslegung und Kostenoptimierung von Leitungsnetzen für Datenfernübertragung, Computer Praxis 6 (1973), S. 133–142.

[3] *Pade, E.:* Ein heuristisches Verfahren für die Kostenoptimierung von Konzentratornetzen. Elektron. Rechenanlagen 17 (1975), S. 212–219.

[4] *Greve, R. H.:* Wählleistung statt Standleitung. Computerwoche 1. August 1975.

[5] *Planck, A.:* Erhöhte Leitungskosten verhindern On-line-Verbund. Computerwoche 24. 1. 1975.

[6] Einführung des Personenkennzeichens in NW; Veröffentlichungen des Innenministers des Landes NW „Zur Information" No. 7, Düsseldorf 1972.

[7] *Föcker, E.:* Automation im Einwohnerwesen – Zielvorstellungen und Verwirklichung. ÖVD 3/73.

[8] *Goller, F., Scheuring, H., Trageser, A.:* Das KI-System. Automatisierte Kommunikation und Information in Politik und Verwaltung. Kohlhammer, Stuttgart 1971.

[9] *Ostermann, J., von Berg, M.:* Automation im Einwohnerwesen – Chance und Bewährung der öffentlichen Verwaltung. ÖVD 9/74.

[10] *Barth, I.:* Vergleich technischer Verfahren für den landesübergreifenden Datenaustausch im PK-Vergabesystem. Bericht des Instituts für Datenfernverarbeitung der GMD, Darmstadt (Jan. 1974).

[11] *Hinsch, E., Randszus, F., Rave, D.:* Untersuchung über Auswirkung des EDS auf die Datenfernverarbeitung im Bereich der öffentlichen Verwaltung. Bericht des Instituts für Datenfernverarbeitung der GMD, Darmstadt (Febr. 1974).

[12] Diebold Management Report, Dezember 1973, S. 2.

[13] *Köhler, W.:* Überlegungen zur Gestaltung eines Netzwerkes für Datenfernverarbeitung. ÖVD 2/76.

[14] *Eckbauer, D.:* Rechnerverbund aus Minisystemen. Computerwoche 25. Juli 1975, S. 14.

[15] *Dropmann, E.:* Verfahren zur Datenübertragungssteuerung. ÖVD 4/75.

[16] *Schlangen, J.:* Die Wirtschaftlichkeit der Datenverarbeitung in einem Versicherungsunternehmen. IBM-Nachrichten 226 (1975), S. 168.

[17] Data Communications in European Industry. The Diebold Research Programm Europe E 118 S (1973), S. 6 und 7.

[18] *Havermann, H., Hinz, D., Schmidt, J. F.:* Die Anwendung von System-Design-Verfahren in einer Datenfernverarbeitungsanwendung. IBM-Nachrichten 217 (1973), S. 793.

[19] Die britische Post startet den „Paketverkehr" zwischen den Computer. Online 1975, S. 387.

[20] Transpac-Netz für Frankreichs Telekommunikation. Computerwoche 6. Februar 1976.

Schleifen und strukturierte Programmierung

Loops and structured programming

Elektron. Rechenanl. 18 (1976), H. 4, S. 172–179
Manuskripteingang: 2. Februar 1976

von J. F. H. WINKLER
Institut für Informatik III
der Universität Karlsruhe

Ein zentrales Problem bei der Konstruktion übersichtlicher und gut handhabbarer Programme mit Hilfe der strukturierten Programmierung ist die Realisierung von Schleifen. Dabei spielt insbesondere die Formulierung von Fehlerausgängen eine wichtige Rolle. In den letzten Jahren wurden zu diesem Problem eine Reihe von Lösungsvorschlägen gemacht, die in der vorliegenden Arbeit an Hand von zwei Programmbeispielen miteinander verglichen werden.

When composing well readable and manageable programs using structured programming a central problem one faces is the realization of loops. Especially important is the formulation of error exits. Several solutions, which were proposed in the last few years, are compared in this paper by means of two program examples.

1. Einleitung

Im Zusammenhang mit der Konstruktion übersichtlicher und gut handhabbarer Programme mit Hilfe der sogenannten strukturierten Programmierung hat sich herausgestellt, daß ein zentrales Problem dabei die Realisierung von Schleifen darstellt. Dies hat zweierlei Gründe: zum ersten kann gezeigt werden, daß die einfachste Methode zur Erzielung wohlstrukturierter Programme, die Knotenspaltung, bei bestimmten Schleifen nicht anwendbar ist [1, S. 410]. Zum zweiten stellen Schleifen ein sehr häufig verwendetes Element von Programmiersprachen dar, da die meisten Anwendungen des Rechners darin bestehen, daß einfache Abläufe wiederholt ausgeführt werden.

Zur Umwandlung beliebiger Programme in wohlstrukturierte, äquivalente (schwach äquivalent im Sinne von [2, S. 241]) Programme sind verschiedene Hilfsmittel vorhanden:

a) Knotenspaltung,
b) Bildung von Prozeduren,
c) Einführung von Hilfsvariablen,
d) Einführung neuer Steueranweisungen.

Als Steueranweisungen werden im folgenden die Anweisungen bezeichnet, die den dynamischen Ablauf eines Programmes abweichend von der Reihenfolge der Aufschreibung steuern (z. B. Fallunterscheidung, Schleife).

Zur Beurteilung der so gewonnenen wohlstrukturierten Programme kommen vier Kriterien in Betracht:

a) Veränderung der statischen Programmlänge,
b) Veränderung der dynamischen Programmlänge,
c) Veränderung der Programmtopologie,
d) Übersichtlichkeit und Lesbarkeit („psychological complexity" im Sinne von *Weissman* [3]).

Dabei beziehen sich die ersten beiden Punkte auf das Objektprogramm (internes Programm), während die letzten beiden Punkte das Quellprogramm (externes Programm) betreffen. Manche Verfahren, wie z. B. [4] oder die Simulation eines Befehlszählers [5, S. 274; 6, S. 67] führen oft zu einer starken Veränderung der Programmtopologie.

Zum letzten Punkt ist zu bemerken, daß Programme, die in einem formalen Sinne wohlstrukturiert sind, der dadurch zum Ausdruck kommt, daß zur Formulierung nur bestimmte Steueranweisungen verwendet werden, durchaus unübersichtlich sein können [5, S. 274; 6, S. 67; 7, S. 505]. Dies hat seinen Grund darin, daß der oben verwendete Begriff der Übersichtlichkeit völlig auf den menschlichen Leser des Programmes bezogen ist. Was für diesen Leser übersichtlich ist, kann jedoch nicht theoretisch abgeleitet, sondern muß empirisch ermittelt werden, wozu auch bereits einige Ansätze zu erkennen sind [3; 8]. Daß die rein theoretischen Untersuchungen über die Mächtigkeit von Mengen von Steueranweisungen [5; 9] nicht automatisch für die Praxis des Programmierens bedeutsame Ergebnisse liefern, wird in [10] an einem Beispiel eindrucksvoll gezeigt.

Die Qualität der durch die Umwandlung erhaltenen Programme hängt nun ganz wesentlich von den zur Formulierung zur Verfügung stehenden Steueranweisungen ab, denn die ersten drei der oben erwähnten Hilfsmittel führen oft zu einer Qualitätsverschlechterung des Programmes im Sinne der oben angegebenen Beurteilungskriterien.

- *Knotenspaltung:* diese führt allgemein zu einer Vergrößerung der statischen Länge.
- *Bildung von Prozeduren:* dadurch kann die statische Länge kleiner werden; die dynamische Länge wird in jedem Falle größer. Auch kann die Lesbarkeit durch das Bilden von Prozeduren schlechter werden [11, S. 277].
- *Hilfsvariable:* dies führt allgemein zu einer Vergrößerung der statischen und der dynamischen Programmlänge. Die Lesbarkeit wird oft schlechter [5, S. 274; 6, S. 67].

Generell wird man damit zu rechnen haben, daß ein besser lesbares und damit besser handhabbares Programm (bezüglich Änderungen etc.) eine etwas geringere Effektivität aufweisen wird als ein möglichst effektives Programm. Man muß dabei aber auch bedenken, daß die Richtigkeit eines Programmes um so besser überprüft werden kann, je besser das Programm lesbar ist.

Die zentrale Bedeutung der zur Verfügung stehenden Steueranweisungen führte in den letzten Jahren zu einer Reihe von Vorschlägen für Sprachkonstruktionen zur Formulierung von Schleifen [6; 12 bis 20]. Diese sollen nun im folgenden kritisch miteinander verglichen werden. Dazu werden zwei kleine Programmbeispiele betrachtet, welche die hauptsächlichen Schwierigkeiten enthalten, nämlich eine Schleife mit mehreren Eingängen und eine Schleife mit mehreren Ausgängen. Damit sind allerdings noch nicht alle Schwierigkeiten erfaßt, weitere werden bei den einzelnen Konstruktionen selbst behandelt. Die betrachteten Beispiele sind als Ablaufgraphen in Bild 1 dargestellt.

P1 ist eine Schleife mit zwei Ausgängen (T1 und T2), und P2 ist eine Schleife mit zwei Eingängen (zwischen B und C und zwischen C und D). Die Struktur von P1 hat in der Praxis große Bedeutung, da mehrfache Ausgänge aus Schleifen in der Form von Fehlerausgängen sehr häufig vorkommen. Die meisten der in den letzten Jahren vorgeschlagenen Sprachkonstruktionen für Schleifen sind daher auf Strukturen in der Art von P1 zugeschnitten. Bei dieser Struktur treten auch die größeren Schwierigkeiten bei der Formulierung als wohlstrukturiertes Programm auf, wenn beispielsweise nur die Steueranweisungen aus [21] zur Verfügung stehen. Da die Knotenspaltung hierzu nicht ausreicht [1; 22; 23], muß man in der Regel zusätzliche Hilfsvariable zur Steuerung des Programmablaufs einführen, wenn man nicht die Methode von *Wijngaarden* [4] zur Eliminierung von Sprüngen mittels Prozeduren anwenden will. Diese Methode führt allerdings oft zu sehr unübersichtlichen Programmen [23, S. 23], und daher soll sie im folgenden nicht verwendet werden. Gegen die übermäßige Verwendung von Prozeduren sind auch von anderer Seite Einwände vorgebracht worden [11, S. 277; 24].

Die Struktur P2 läßt sich mittels einfacher Schleifenanweisungen durch die Knotenspaltung realisieren [7, S. 507]. Bei der Schaffung von besonderen Sprachkonstruktionen für Strukturen der Art P2 kann es nur darum gehen, den durch die Knotenspaltung zusätzlichen Speicheraufwand zu verringern oder zu vermeiden, da die dynamische Länge unverändert bleibt. Darauf wird im Abschnitt 10 näher eingegangen.

Für die folgenden Formulierungen mittels der behandelten Sprachkonstruktionen gelten folgende Rahmenbedingungen:

a) Es werden stets anweisungsorientierte Sprachen (im Sinne von Algol 60) angenommen. Der Grund dafür ist, daß ausdrucksorientierte Sprachen wie Algol 68 [25] und Bliss [12; 26] bei Ausnutzung ihrer Möglichkeiten leicht zu sehr unübersichtlichen Programmen führen können [1, S. 412; 15, S. 11, 13; 27, S. 238].

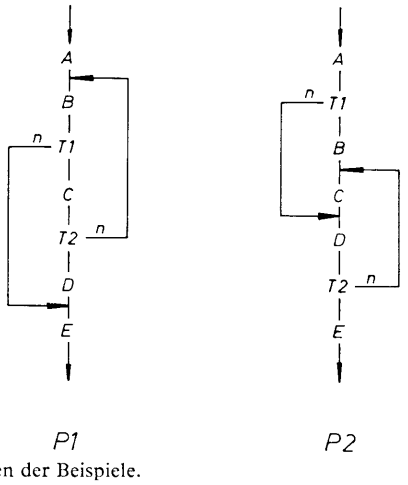

Bild 1. Die Ablaufgraphen der Beispiele.

b) Die bedingte Anweisung
if ⟨Bed⟩ then ⟨Anw-Folge⟩ [else ⟨Anw-Folge⟩] fi
ist stets zugelassen.

c) Prozeduren sollen nicht neu gebildet werden.

d) Wo keine konkrete Syntax gegeben ist, werden übliche Notierungen verwendet.

e) goto soll nicht verwendet werden, auch wenn es in den betrachteten Sprachen (z. B. Algol 68) enthalten ist.

2. if, while, until

Bei *Dijkstra* [21] sind folgende Steueranweisungen zugelassen: if ... then ... else ..., case, while ... do ..., repeat ... until .... Wie schon *Wulf* [1, S. 410] gezeigt hat, läßt sich P1 mit dem vorgegebenen Satz von Steueranweisungen nur mit Hilfsvariablen realisieren. Es ergibt sich etwa folgendes Programm:

```
A;
BB := true;
while BB do
begin B;
        if not T1
        then BB := false
        else C;
             if T2 then D; BB := false; fi
        fi
end
E;
```

P2 kann wahlweise mit Knotenspaltung oder mit Hilfsvariablen realisiert werden. Knotenspaltung ergibt:

```
A;
if T1 then B; C; fi
D;
while not T2 do
begin C; D; end
E;
```

Hierbei müssen C und D je zweimal notiert werden.
Mit Hilfsvariablen ergibt sich:

```
A; BB := false;
if T1 then B; else BB := true; fi
repeat begin
        if not BB then C; fi
        BB := false;
        D;
        end
until T2;
E;
```

3. while-until-Schleife

Die in [15] vorgeschlagene while-until-Schleife bringt bei der Formulierung von P1 und P2 keine Vorteile, denn diese Schleife ist im wesentlichen äquivalent zu einer Schleife mit dem Test am Schleifenanfang:

```
while T1 repeat S until T2
```

ist äquivalent zu

```
if T1 then S; while not T2 and T1 do S; fi
```

Nach dem ersten Durchlauf verhält sich die Schleife wie eine einfache while-Schleife. Es ergeben sich daher im wesentlichen Formulierungen wie in Abschnitt 2.

```
P1:  A; BB := true;
     while BB
     repeat begin B;
                 if not T1
                 then BB := false;
                 else C;
                      if T2
                      then D; BB := false;
                      fi
                 fi
          end
     until false;
     E;

P2:  A;
     if T1 then B; C; fi
     D;
     while not T2
     repeat begin C; D; end
     until false;
     E;
```

4. Einstufige Fluchtanweisung

Der wesentliche Grund für die Schwierigkeiten mit den in den Abschnitten 2 und 3 verwendeten Sprachmitteln ist der, daß bei ihnen das Verlassen einer Schleife an beliebiger Stelle über Hilfsvariable gesteuert werden muß. Um ein solches Verlassen zu ermöglichen, wurden die Fluchtanweisungen geschaffen.

Die Fluchtanweisung als Algol 68, der sogenannte completer [25, S. 54], ist nur im Zusammenhang mit dem goto sinnvoll verwendbar. Daher auch die Vorschrift in [25], daß das auf das exit folgende Element der seriellen Klausel mit einer Marke versehen sein muß. Da das goto hier nicht zugelassen sein soll, lassen sich mit Algol 68 nur die Lösungen aus Abschnitt 2 formulieren, da auch die Ausdrucksorientiertheit nicht verwendet werden sollte. Eine bessere einstufige Fluchtanweisung stellt die break-Anweisung in BCPL [14; 28] dar. Allgemein läßt sich festhalten, daß die einstufige Fluchtanweisung beim Verlassen mehrerer geschachtelter Schleifen Hilfsvariable erfordert [2; 7]. Für P1 ergibt sich in BCPL:

```
A;
$( B;
   if not T1 do break;
   C;
   if T2 do $( D; break $)
$) repeat;
E;
```

Die Beschränkung der break-Anweisung auf eine Stufe macht sich bei P1 noch nicht bemerkbar; man sieht die Beschränkung jedoch leicht an Beispielen mit mehreren geschachtelten Schleifen.

Da die break-Anweisung ein Verlassen einer Schleife an beliebiger Stelle ermöglicht, ergibt sich bei P2 eine etwas effektivere Lösung:

```
A;
if T1 do $( B; C $);
$( D;
   if T2 do break;
   C;
$) repeat;
E;
```

In dieser Realisierung ist nur noch C verdoppelt.

5. Mehrstufige Fluchtanweisung

Bei der mehrstufigen Fluchtanweisung wie etwa in Bliss [6, S. 68] werden die Schleifen mit Marken versehen, und eine Anweisung „leave A" führt zum Verlassen der mit A markierten Schleife:

```
P1: A;
    SCHL: repeat B;
          if not T1 then leave SCHL; fi
          C;
          if T2 then D; leave SCHL; fi
          until false;
    E;

P2: A;
    if T1 then B; C; fi
    SCHL: while true
          do D;
             if T2 then leave SCHL; fi
             C;
          od
    E;
```

Bei beiden Realisierungen stellt man fest, daß der eigentliche Schleifentest völlig überflüssig ist, und die Steuerung nur durch die Fluchtanweisung erfolgt.

6. Der Vorschlag von Bochmann

Der Vorschlag von *Bochmann* [13] berücksichtigt die Feststellung, daß oft der Fall vorliegt, daß nach dem Verlassen der Schleife von verschiedenen Punkten aus, verschiedene Aktionen durchgeführt werden sollen, wie dies bei P1 der Fall ist. *Bochmann* schlägt eine einstufige Fluchtanweisung mit einer Marke vor (exitloop ⟨Marke⟩), wobei diese Marke eine Anweisung nach der Schleife markiert, die nach dieser Fluchtanweisung durchgeführt werden soll. Anschließend wird die Ausführung nach der letzten markierten Anweisung, die zu der betreffenden Schleife gehört, fortgesetzt. Wird die Schleife durch den Schleifentest abgebrochen, dann wird die durch die Standardmarke „ended:" markierte Anweisung ausgeführt. In Verallgemeinerung zu [13], wo diese Fluchtanweisung nur im Zusammenhang mit Zählschleifen definiert wird, wird sie im folgenden auch mit while- und until-Schleifen verwendet. Damit ergeben sich für P1 und P2 folgende Formulierungen:

```
P1: A;
    repeat B;
          if not T1 then exitloop MM; fi
          C;
    until T2;
    ended: D;
    MM: ;
    E;

P2: A;
    if T1 then B; C; fi
    while true
    do D;
       if T2 then exitloop MM; fi
       C;
    od
    ended: ;
    MM: E;
```

Das Durchführen verschiedener Aktionen nach dem Verlassen einer Schleife von verschiedenen Stellen läßt sich auch mittels der in den Abschnitten 4 und 5 vorgestellten Fluchtanweisungen realisieren, wie die dortigen Formulierungen von P1 zeigen. Steht nach der Marke MM: die Anweisung A, dann kann statt

```
if B then exitloop MM; fi
```

in BCPL dasselbe durch

```
if B do S( A; break S);
```

erreicht werden.

Bei dem vorliegenden Vorschlag von *Bochmann* tritt zusätzlich das Problem auf, daß die Aktion, die nach dem Verlassen der Schleife durchgeführt werden soll, an einer anderen Stelle steht als die Fluchtanweisung selbst. Es ergibt sich ein ähnliches Zuordnungsproblem wie in Algol 60 bei der Deklaration einer Prozedur, wo die Spezifikation und Parameterbezeichnung und die Übergabeart in verschiedenen Listen stehen. Dieses Zuordnungsproblem tritt bei den Konstruktionen in Abschnitt 3 und 4 nicht auf. Da die Fluchtanweisung aus [13] auf eine Stufe beschränkt ist, ergeben sich beim Verlassen mehrerer geschachtelter Schleifen Unbequemlichkeiten: es muß eine Folge von exitloop-Anweisungen durchlaufen werden.

7. Der Vorschlag von Zahn

Der Vorschlag von *Zahn* [20] stimmt im wesentlichen mit dem Vorschlag von *Bochmann* überein, indem er eine Lösung für das unterschiedliche Fortsetzen nach dem Verlassen der Schleife an verschiedenen Punkten ermöglicht. Lediglich die konkrete syntaktische Form ist etwas anders. Anstelle der Marken treten Ereignisbezeichner, die in einer Liste am Anfang der Schleife in Form einer Disjunktion zusammengefaßt sind. Die Ereignisbezeichner kann man sich als logische Variable vorstellen, die vor Beginn der Schleifenabarbeitung alle den Wert „false" haben. Innerhalb des Schleifenkörpers führt das Auftreten eines solchen Bezeichners dazu, daß diese logische Variable auf „true" gesetzt wird und sofort die erwähnte Disjunktion ausgewertet wird. Die Schleife wird solange wiederholt, wie die Disjunktion den Wert false ergibt. Im Anschluß an den Schleifenkörper folgt eine Folge von Anwei-

sungen, die mit je einem der Ereignisbezeichner markiert sind. Dadurch wird auf genau dieselbe Art und Weise wie bei *Bochmann* [13] die Folgeaktion nach Verlassen der Schleife bestimmt. Die Marken bei *Bochmann* [13] entsprechen daher den Ereignissen bei *Zahn* [20], wobei es bei Zahn kein Pendant zu der Standardmarke „ended" gibt. Für P1 und P2 ergeben sich im wesentlichen dieselben Programme, die sich nur in der konkreten Syntax unterscheiden:

```
P1:  A;
     until E1 or E2 do
     begin B; if not T1
           then E1
           else C; if T2 then E2 fi
           fi
     end
     then case
     E1: ;
     E2: D;
     E;

P2:  A;
     if T1 then B; C; fi
     until E1 do
     begin D;
           if T2 then E1 else C; fi
     end
     E1: E;
```

Da es sich im wesentlichen um eine einstufige Fluchtanweisung handelt, ist das Verlassen mehrerer geschachtelter Schleifen etwas unbequem zu formulieren [20, S. 174]. Außerdem ergibt sich auch hier das in Abschnitt 6 erwähnte Zuordnungsproblem.

8. Der Vorschlag von Halaas

Ausgehend von der Konstruktion von *Zahn* [20] hat *Halaas* kürzlich eine weitere ereignisgesteuerte Konstruktion vorgeschlagen [16]. Der wesentliche Unterschied zum Vorschlag von *Zahn* besteht darin, daß die Folge der auf den Schleifenkörper folgenden Anweisungen in den Schleifenkörper selbst einbezogen wird, indem das Auftreten eines Ereignisbezeichners im Schleifenkörper ersetzt wird durch die zugehörige markierte Anweisung. Damit tritt das in Abschnitt 6 erwähnte Zuordnungsproblem nicht mehr auf. Falls das gleiche Ereignis mehrfach auftritt, dann muß die zugehörige Anweisung, welche beim Verlassen der Schleife ausgeführt werden soll, mehrfach notiert oder als Prozedur formuliert werden. Das Verlassen mehrerer geschachtelter Schleifen wird dadurch erleichtert, daß die vorgeschlagene Sprachkonstruktion rekursiv ist, und daß in weiter innen liegenden Schleifen auch die Ereignisbezeichner der äußeren Schleifen gültig sind, sofern keine Benennungskonflikte auftreten. Zur konkreten Syntax ist zu bemerken, daß die Disjunktion der Ereignisbezeichner durch das Wortsymbol „detect" eingeleitet und der Schleifenkörper durch die Wortsymbole „in" und „ni" eingeschlossen wird. Die Anweisung, die beim Verlassen der Schleife infolge des Auftretens eines Ereignisses ausgeführt wird, ist in [16] in geschweifte Klammern eingeschlossen; dafür werden hier „⟨(" und „)⟩" verwendet.

Für P1 und P2 ergeben sich damit die folgenden Programme:

```
P1:  A;
     detect VERLASSE
     in B;
        if not T1 then VERLASSE; fi
        C;
        if T2 then VERLASSE ⟨(D)⟩; fi
     ni
     E;

P2:  A;
     if T1 then B; C; fi
     detect VERLASSE
     in D;
        if T2 then VERLASSE; fi
        C;
     ni
     E;
```

9. Der Vorschlag von Martin

Aufgrund graphentheoretischer und kombinatorischer Überlegungen kommt *Martin* in [17] zu der in Bild 2 dargestellten Menge von Programmbausteinen [17, S. 11], die von ihm als „minimal ‚natural' set of basic control structures" bezeichnet werden [17, S. 10].

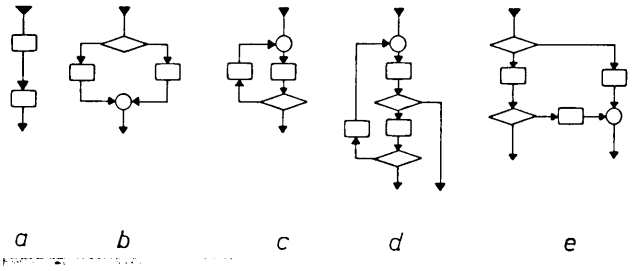

Bild 2. Die „natürliche" Mindestmenge von Steueranweisungen nach *Martin* [17].

Kompliziertere Graphen können erhalten werden, indem ein Knoten mit einem Eingang und einem Ausgang durch einen der Bausteine a, b, c ersetzt wird, oder ein Knoten mit einem Eingang und zwei Ausgängen durch einen der Bausteine d oder e ersetzt wird. Alle Graphen, die man durch endlich viele solcher Ersetzungen aus dem primitiven Graphen, der aus einem Knoten mit einem Eingang und einem Ausgang besteht, erzeugen kann, sind in seinem Sinne zulässige Flußgraphen („flow-graphs" [17, S. 10]). Da *Martin* selbst keine Syntax für seine Bausteine angibt, wird im folgenden lediglich dargestellt, durch welche Ersetzungen P1 und P2 hergeleitet werden können, wobei davon ausgegangen wird, daß manche (1,1)-Knoten auch die leere Anweisung repräsentieren können. Für P1 ist diese Herleitung in Bild 3 dargestellt. Der Knoten, der in einem Ersetzungsschritt ersetzt wird, ist jeweils schraffiert.

P2 ist in der vorliegenden Form nicht unmittelbar realisierbar, da P2 nicht reduzierbar im Sinne von [29] ist, während man zeigen kann, daß alle nach obiger Vorschrift aus den fünf Bausteinen erzeugbaren Flußgraphen in diesem Sinne reduzierbar sind. Die in den vorangehenden Abschnitten

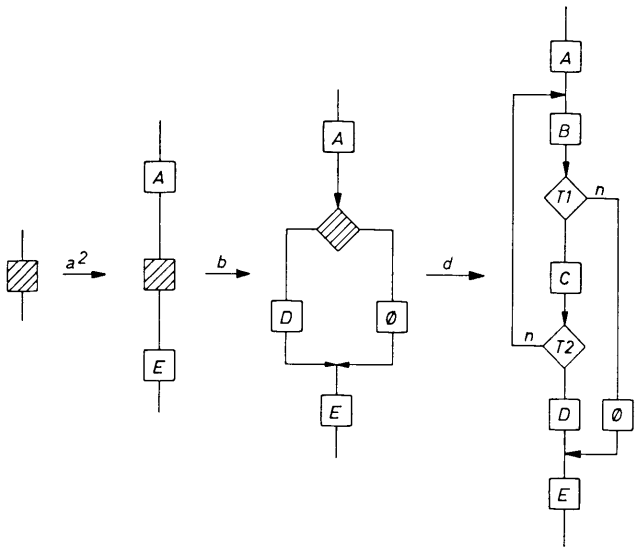

Bild 3. Realisierung von P1 mit den Bausteinen von *Martin* [17].

durch Knotenspaltung erhaltene Formulierung von P2 ist jedoch realisierbar und in Bild 4 dargestellt.

10. Eigener Vorschlag

Aus den Untersuchungen in den vorangehenden Abschnitten ergeben sich folgende Anforderungen an eine Sprachkonstruktion zur Formulierung von Schleifen:

a) Es muß möglich sein, eine Schleife an einer beliebigen Stelle des Schleifenkörpers zu verlassen.

b) Das Verlassen der Schleife muß an beliebig vielen Stellen des Schleifenkörpers möglich sein.

c) Es muß das Verlassen beliebig vieler geschachtelter Schleifen möglich sein.

Außerdem kann man folgende Feststellungen treffen:

a) Beim Vorhandensein einer Fluchtanweisung ist eine eigene Schleifenbedingung oft überflüssig (s. Abschnitt 5).

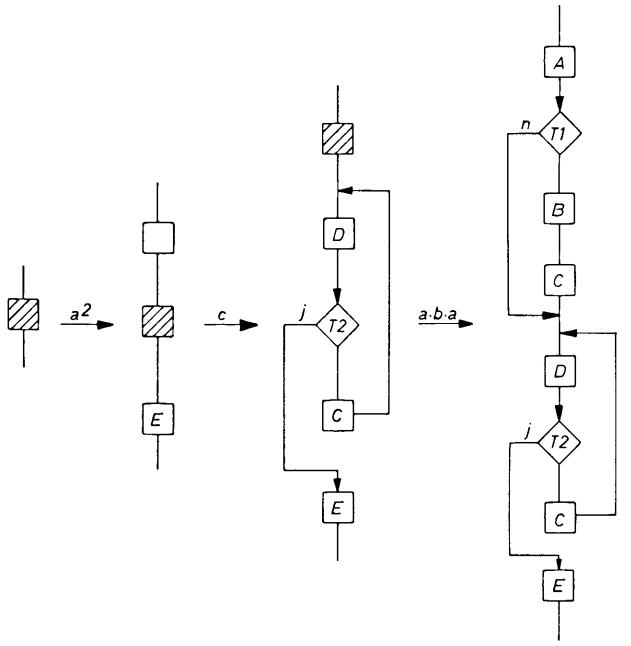

Bild 4. Realisierung von P2 mit den Bausteinen von *Martin* [17].

b) Schleifen mit mehreren Eingängen kommen in der Praxis nicht so häufig vor wie Schleifen mit mehreren Ausgängen. Die ausführliche Arbeit von *Knuth* [5] enthält beispielsweise kein einziges Beispiel einer Schleife mit mehreren Eingängen.

Diese Anforderungen und Feststellungen führten zu der folgenden Sprachkonstruktion, die bereits in einer Sprache zur Generierung von Programmen verwendet und auch implementiert worden ist [30].

⟨Schleife⟩ ::= rep ⟨Bez⟩ ⟨Anw-Folge⟩ per ⟨Bez⟩
⟨Flucht-Anw⟩ ::= leave ⟨Bez⟩ [when ⟨Log-Ausdr⟩];

Bei dieser Konstruktion sind Wiederholung und Verlassen der Schleife jeweils durch ein Sprachelement ausgedrückt und durch Kombination beider ergeben sich die üblichen Schleifentypen. Die Form der Fluchtanweisung mit Bedingung kann als Abkürzung für

if ⟨Log-Ausdr⟩ then leave ⟨Bez⟩; fi

aufgefaßt werden.

Für P1 und P2 ergeben sich folgende Programme:

```
P1:  A;
     rep SCHL
        B;
        leave SCHL when not T1;
        C;
        if T2 then D; leave SCHL; fi
     per SCHL
     E;
```

Da diese Konstruktion auf Schleifen mit mehreren Ausgängen zugeschnitten ist, ergibt sich für P2 nur die bereits in den vorangehenden Abschnitten dargestellte Lösung, bei der C verdoppelt wird.

```
P2:  A;
     if T1 then B; C; fi
     rep SCHL
        D;
        leave SCHL when T2;
        C;
     per SCHL
     E;
```

Um auch für Strukturen von der Art von P2 eine effektivere Lösung zu ermöglichen, müßte man eine Schleifenkonstruktion vorsehen, die es erlaubt, die Abarbeitung der Schleife an einem beliebigen, etwa durch eine Marke markierten, Punkt des Schleifenkörpers zu beginnen. Da dadurch selbst keine Schleifen gebildet werden sollen, ist diese Schleifeninitiierung nur in Form eines Vorwärtssprunges nötig. Eine solche Form der Schleifeninitiierung kann man als Eintrittsanweisung bezeichnen und als ein Gegenstück zur Fluchtanweisung auffassen. Ein Vorschlag hierzu findet sich in Diagrammform in [18, S. 43]. Eine konkrete Syntax dazu könnte etwa folgendermaßen festgelegt werden:

```
⟨Schleife⟩      ::= loop ⟨Bez⟩ ⟨Anw-Folge-1⟩
                    rep ⟨Bez⟩ ⟨Anw-Folge-2⟩
                    per ⟨Bez⟩
```

wobei noch folgende Regeln gelten sollen:

```
⟨Anweisung⟩      ::= ⟨Marke⟩ ⟨Unmark-Anw⟩
⟨Unmark-Anw⟩     ::= enter ⟨Bez⟩ at ⟨Bez⟩ ; /
                     leave ⟨Bez⟩ ; / ...
```

Damit läßt sich P2 folgendermaßen realisieren:

```
    A;
    loop SCHL
        if not T1 then enter SCHL at MM; fi
        B;
    rep SCHL
        C;
MM: D;
        if T2 then leave SCHL; fi
    per SCHL
    E;
```

Die Einhaltung der oben erwähnten Bedingung, daß die Eintrittsanweisung nur in Vorwärtsrichtung führen darf, und die Bedingung, daß durch die Ausführung der Eintrittsanweisung keine Vereinbarung übersprungen werden darf, sowie das Problem des Eintritts über mehrere Schachtelungsstufen hinweg, führen dazu, daß man Zusatzregeln darüber aufstellen muß, welche Markenbezeichner in der Eintrittsanweisung zulässig sind.

Solche Restriktionen sind:

a) In einer Eintrittsanweisung, die Teil der ⟨Anw-Folge-1⟩ ist, darf nur eine Marke verwendet werden, die eine Stelle in der ⟨Anw-Folge-2⟩ markiert.

b) Eintrittsanweisung und Ziel der Eintrittsanweisung müssen in ein und demselben Gültigkeitsbereich liegen.

11. Zusammenfassung

An einer Zahl verschiedener Sprachkonstruktionen für Schleifen wurde gezeigt, daß die Möglichkeit zur Komposition übersichtlicher, gut lesbarer und möglichst auch noch effektiver Programme von den zur Verfügung stehenden Steueranweisungen abhängt. Dabei zeigt sich allgemein, daß eine Beschränkung auf eine möglichst geringe Anzahl (Minimalmenge) von Steueranweisungen in der Regel dazu führt, daß die damit formulierten Programme lang und unübersichtlich werden und insbesondere eine hohe Komplexität aufweisen. Als Beispiel dafür kann auf die erste Form der Schleife in Abschnitt 10 hingewiesen werden. Dort wurde schon bemerkt, daß die Fluchtanweisung mit Bedingung auch aus einer unbedingten Fluchtanweisung und einer bedingten Anweisung zusammengesetzt werden kann. Diese Lösung führt aber zu einem längeren Quellprogramm. Will man z. B. nur eine einfache while-Schleife realisieren, dann führt die Konstruktion aus Abschnitt 10 ebenfalls zu einer etwas schwerfälligen Lösung:

```
while T do S; od
```

muß formuliert werden als:

```
rep SCHL
    leave SCHL when not T;
    S;
per SCHL
```

Der Vorschlag in Abschnitt 10 soll daher nicht bedeuten, daß nun alle Schleifen in einem Programm mit dieser Konstruktion formuliert werden sollen, sondern daß zur Formulierung von Schleifen, die mit den einfachen while- oder until-Konstruktionen nicht bequem ausgedrückt werden können, eine geeignete Sprachkonstruktion bereitgestellt wird. Auf die Bedeutung von Schleifen mit mehreren Ausgängen wurde bereits in der Einleitung hingewiesen. Eine Programmiersprache sollte daher nicht nur einen Minimalsatz von Steueranweisungen enthalten. Ein Beispiel für einen umfangreichen Satz von Schleifenkonstruktionen bietet die Sprache BCPL [28].

Da man niemanden daran hindern kann, mit Absicht unübersichtlich zu programmieren, sollte man wenigstens den Gutwilligen geeignete Sprachkonstruktionen zur Verfügung stellen, die eine übersichtliche und effektive Formulierung ihrer Probleme erlauben.

Literatur

[1] *Wulf, William A.:* Programming without the GOTO. Information Processing 71. North Holland Publ. Comp. Amsterdam 1972, S. 408–413.

[2] *Kosaraju, Rao S.:* Analysis of structured Programms. 5. Annual ACM Symposium on Theory of Computing. ACM 1973.

[3] *Weissman, Laurence M.:* A Methodology For Studying The Psychological Complexity of Computer Programs. Thesis Ph. D., Univ. of Toronto CSRG-37, August 1974.

[4] *Wijngaarden, A. van:* Recursive Definition of Syntax and Semantics. = [32, S. 13–24].

[5] *Knuth, Donald E.:* Structured Programming with goto Statements. Computing Surveys 6 (1974), 4, S. 261–301.

[6] *Wulf, William A.:* A Case Against the GOTO. SIGPLAN Notices 7 (1972), 11, S. 63–69.

[7] *Peterson, W. W.; Kasami, T.; Tokura, N.:* On the Capabilities of While, Repeat, and Exit Statements. CACM 16 (1973), 8, S. 503–512.

[8] *Gannon, J. D.; Horning, J. J.:* The Impact of Language Design on the Production of Reliable Software. SIGPLAN Notices 10 (1975), 6, S. 10–22.

[9] *Boehm, Corrado; Jacopini, Giuseppe:* Flow Diagrams, Turing Machines and Languages with only two Formation Rules. CACM 9 (1966), 5, S. 366–371.

[10] *Ledgard, Henry F.; Marcotty, Michael:* A Genealogy of Control Structures. CACM 18 (1975), 11, S. 629–639.

[11] *Pager, David:* On the Problem of Communicating Complex Information. CACM 16 (1973), 5, S. 275–281.

[12] *Wulf, W. A.; Russel, D.; Habermann, A. N.; Geschke, C.; Apperson, J.; Wile, D.; Brender, R.:* Bliss Reference Manual. Department of Computer Science. Carnegie Mellon Univ. 15. 01. 1970.

[13] *Bochmann, G. V.:* Multiple Exits from a Loop Without the GOTO. CACM 16 (1973), 7, S. 443–444.

[14] *Evans, R. V.:* Multiple Exits from a Loop Using Neither GOTO nor Labels. CACM 17 (1974), 11, S. 650.

[15] *Friedman, Daniel P.; Shapiro, Stuart C.:* A Case for while-until. SIGPLAN Notices 9 (1974), 7, S. 7–14.

[16] *Halaas, Arne:* Event-driven control statements. BIT 15 (1975), 3, S. 259–271.

[17] *Martin, Johannes J.:* The „Natural" Set of Basic Control Structures. SIGPLAN Notices 8 (1973), 12, S. 5–14.

[18] *Wegner, Eberhard:* Baumstrukturierte Programme. Diss. Techn. Univ. Berlin 1974.

[19] *Wilner, W. T.:* Structured Programms, Arcadian Machines, and the Burroughs B1700. Lecture Notes in Computer Science 7. Springer Verlag, Berlin usw. 1974, S. 133–148.

[20] *Zahn, Charles T. J.:* A Control Statement For Natural Top-Down Structured Programming. = [33, S. 170–180].

[21] *Dijkstra, Edsger W.:* Notes on Structured Programming = [31, S. 1–82].
[22] *Ashcroft, Edward; Manna, Zohar:* The Translation of „goto" Programs to „while" Programs. Information Processing 71. North Holland Publ. Comp. Amsterdam 1972, S. 250–255.
[23] *Knuth, D. E.; Floyd, R. W.:* Notes on avoiding „goto" Statements. Inf. Proc. Letters 1 (1971), S. 23–31.
[24] *Abrahams, Paul:* „Structured Programming" Considered Harmful. SIGPLAN Notices 10 (1975), 4, S. 13–24.
[25] *Wijngaarden, A. van; Mailloux, B. J.; Peck, J. E. L.; Koster, C. H. A.; Sintzoff, M.; Lindsay, C. H.; Meertens, L. G. L. T.; Fisker, R. G.* (eds): Revised Report on the Algorithmic Language ALGOL 68. Springer Verlag, Berlin usw. 1976.
[26] *Wulf, W. A.; Russel, D. B.; Habermann, A. N.:* BLISS: A Language for Systems Programming. CACM 14 (1971), S. 780–790.
[27] *Desjardins, P.; Lecarme, O.:* More Comments on the Programming Language Pascal. Acta Informatica 4 (1975), S. 231–243.
[28] *Richards, Martin:* BCPL: A tool for compiler writing and system programming. SJCC 1969, S. 557–566.
[29] *Hecht, Matthew S.; Ullman, Jeffrey D.:* Flow Graph Reducibility. SIAM J. Comp. 1 (1972), 2, S. 188–202.
[30] *Wettstein, H.; Becker-Weimann, K.; Winkler, J. F. H.; Wosnitza, H.:* Adaptive Betriebssystemgenerierung. Projektbericht. UNI KA, IFI III, April 1976.
[31] *Dahl, O.-J.; Dijkstra, E. W.; Hoare, C. A. R.:* Structured Programming. Academic Press, London etc. 1972.
[32] *Steel, T. B. Jr.:* Formal Language Description Languages for Computer Programming. North Holland Publ. Comp. Amsterdam 1966.
[33] *Robinet, B.* (ed): Programming Symposium. Springer Verlag, Berlin usw. 1974.

Einlagerung von Synchronisationsoperationen in Prozeßsysteme

Utilization of synchronizing operations in process systems

Elektron. Rechenanl. 18 (1976), H. 4, S. 179–185
Manuskripteingang: 10. Januar 1976

von H. WETTSTEIN
Institut für Informatik III (Betriebssysteme)
der Universität Karlsruhe

Mit steigendem Parallelitätsgrad spielen Synchronisationsoperationen in Betriebssystemen eine zunehmende Rolle. Um so dringlicher wird deren einheitliche Strukturierung und effiziente Implementierung. Letztere ist u. a. abhängig von der Adressierungsumgebung, in die ein Prozeßsystem eingebettet ist. Diese Arbeit gibt einen Überblick über die verschiedenen Möglichkeiten, angefangen von einer Adreßbindung zur Programmerstellzeit bis zu einem symbolisch-assoziativen Adressierungsmechanismus.

With increasing degree of parallelity synchronizing functions play a growing part in operating systems. Therefore, their unified structuring and efficient implementation becomes more and more urgent. The latter is among other things dependent upon the addressing environment, in which the process system is embedded. This paper gives an overview on the various possibilities starting from an address binding at program compilation time up to a symbolic-associative addressing mechanism.

1. Einleitung

Die in einer Datenverarbeitungsanlage ablaufenden und vom Betriebssystem verwalteten Aktivitäten sind als sogen. parallele Prozesse organisiert. Die moderne Betriebssystemstrukturierung zielt darauf ab, mehr und mehr Teilfunktionen als selbständige Prozesse zu organisieren. Beispiele hierfür sind etwa die Ein/Ausgabebetreiber, die Seitentauschroutinen die Speicher- und die Geräteverwaltung u. ä. Der Trend wird dann noch verstärkt, wenn es gelingt, in den kommenden Jahren die prognostizierte Anhäufung von Mikroprozessoren verschiedener Leistung in der Zentrale der DV-Anlage zu verwirklichen [12]. Es wird dann sinnvoll, schwierige Algorithmen, die stoßweise Leistung beanspruchen, in eine Art „Fortschreibe"-Algorithmen umzuwandeln und diese parallel rechnen zu lassen. Für den Fall der Verklemmungsvermeidung ist in [2] ein Beispiel einer solchen Vorgehensweise zu finden.

Je stärker die Aufgaben eines Betriebssystems in Prozesse gegliedert sind, desto intensiver müssen diese ihre Tätigkeit untereinander abstimmen. Sie müssen sich mit Ereignissen aus anderen Prozessen synchronisieren.

Zu diesem Zweck sind in die Prozeßabläufe Synchronisationsoperationen eingelagert. Sie treten stets als Paar (W, S) auf, wobei W im wesentlichen eine Warte-, S eine zugeordnete Lösefunktion enthält. Gelegentlich wird mit letzterer auch die Vorstellung eines Signalaussendens verbunden. Die beiden Operationen werden demgemäß mit WARTE und SENDE bezeichnet.

Synchronisationen dienen dazu, zeitliche Relationen zwischen Prozessen bzw. Prozeßteilen herzustellen. Da Prozesse aber zeitunabhängig sind, haben Synchronisationen auf die Prozesse als solche keinen Einfluß. In den meisten Fällen ist aber mit der Synchronisation eine Kommunikation in Form