PROGRAMMIERSPRACHEN

Eine vergleichende Studie

B. HIGMAN

LEIPZIG

BSB B. G. TEUBNER VERLAGSGESELLSCHAFT

1971

Format *x.xx.xx*, so daß die Auswahl des Formats sich erst nach der Identifizierung der ersten Ziffer ergibt.

Es sind auch Maschinen mit variabler Befehlslänge bekannt; das ist gewöhnlich dann der Fall, wenn die natürliche Wortlänge nicht für alle Zwecke ausreicht, während andererseits der Gehalt der durch den Befehl zu übermittelnden Information variiert. Das Verfahren zur Feststellung, daß eine gegebene Zeichenfolge unvollständig ist und der Erweiterung bedarf, kann sowohl rekursiv als auch nichtrekursiv sein. Es gibt eine Maschine, deren Grammatik, wie sie vom Programmierer gesehen wird, sich etwa in der Form

$$\langle\text{Befehl}\rangle ::= \langle\text{Funktionswort}\rangle|\langle\text{Befehl}\rangle\ \langle\text{Parameterwort}\rangle$$

schreiben läßt, wobei die Anzahl der Parameterworte durch die Funktion bestimmt wird. Ziffernorientierte Maschinen können auf einem ähnlichen Prinzip oder auf der inversen polnischen Notation beruhen.

9.4. Die ersten Autocodes

Man verläßt das Gebiet der symbolischen Assemblersprachen und begibt sich in den Bereich der Autocodes, sobald man a) dem Compiler die Verantwortung für die Zuweisung der Speicherplätze überträgt und b) auf die umkehrbar eindeutige Zuordnung zwischen den Funktionen der Maschine und den in der Sprache zulässigen Funktionen etwa zugunsten einer Bezeichnungsweise verzichtet, die sich der üblichen mathematischen Schreibweise ein wenig nähert. Auf dieser Stufe ist jedoch der Freiheitsgrad für die Anwendung der mathematischen Schreibweise gewöhnlich fest, wenn nicht sogar sehr eng begrenzt. Der Name Autocode bezeichnete ursprünglich eine Sprache, die für den Computer Mercury (und für einen Vorgänger dieser Sprache, bekannt als Manchester-Autocode) bestimmt war. Im allgemeinen wird jedoch diese Bezeichnung auf eine Anzahl ähnlicher Sprachen auch für andere Maschinen ausgedehnt, und als Bezeichnung für eine Sprachenstufe umfaßt sie zumindest die frühesten Formen des Fortran. Die zuletzt genannte Sprache, deren Name aus „FORmula TRANslator" abgeleitet ist, war ursprünglich eine amerikanische Entwicklung für IBM-Maschinen. Die am weitesten verbreitete Form ist wahrscheinlich das Fortran II (Rabinowitz, 1962), und unsere Ausführungen beziehen sich, wenn nicht ausdrücklich das Gegenteil betont wird, auf diese Version. Fortran IV hat von der Entwicklung des Algol profitiert und beinhaltet viele Eigenschaften, die wir bei der Betrachtung der Algol-Sprache beschreiben wollen.

Bei der Anwendung des Manchester-Autocodes (Brooker, 1956) war der Programmierer gezwungen, einen Ausdruck der Form

$$x = a + b \times c + d \times e \tag{1}$$

auf die Folge

$$\begin{aligned} v1 &= b \times c \\ v2 &= d \times e \\ v1 &= a + v1 \\ x &= v1 + v2 \end{aligned} \tag{2}$$

zurückzuführen, jedoch war er aller Mühe enthoben, sich mit den echten Maschinenadressen oder mit solchen Angelegenheiten zu beschäftigen, wie etwa das anfängliche Löschen der Akkumulatoren usw., wofür er die Kenntnis des speziellen Maschinencodes benötigt hätte. Es waren auch in gewissen Grenzen Möglichkeiten für Anweisungen der Form

$$v2 = f(v1)$$

vorgesehen, z.B. in der Bedeutung $f =$ *Quadratwurzel*. Im Mercury-Autocode und in Fortran war die Zurückführung von (1) auf (2) ebenfalls mechanisiert worden, allerdings mit gewissen Einschränkungen, die sich grundsätzlich aus der Tatsache ergeben haben, daß zur Zeit ihrer Bekanntgabe die Programmiertechnik, die zur Implementation rekursiv definierter Ausdrücke benötigt wird, noch nicht entwickelt worden war. Es wurden auch Einschränkungen akzeptiert, die sich aus der Struktur der Maschinen ergaben, für die sie entwickelt wurden (vgl. hierzu das Zitat aus Brooker und Morris weiter unten). Das Auftauchen von Programmierungsverfahren dieser Art brachte eine derartige Erleichterung gegenüber der Programmierung in Maschinensprachen (sogar schon bei der Anwendung der symbolischen Assemblertechnik), so daß die durch diese Sprachen gesetzten Schranken erst wesentlich später bemerkt wurden. Zu der Zeit erwuchs ein großes Interesse seitens der Anwender, die Programme und Programmteile untereinander austauschten. Als Folgeerscheinung ergab sich eine Tendenz zur Erweiterung beider Sprachen durch Hinzunahme solcher zusätzlichen Elemente, die ohne die Grundelemente in ihren *Originalformen* zu beeinträchtigen, hinzugefügt werden konnten.

Im ursprünglichen Mercury-Autocode standen z.B. die Buchstaben $i, j, \ldots, t$ als ganzzahlige Variable und die anderen als reelle Variable zur Verfügung, und diese Starrheit ist nie beseitigt worden. Fortran gewährte eine größere Freizügigkeit insofern, als Worte aus mehreren Buchstaben als Namen für Variable verwendet werden

konnten, doch spielten die Anfangsbuchstaben jedes Namens eine ähnliche Rolle (*I* bis *N* für ganzzahlige Größen z. B.), indem sie den Typ des Namens festlegten. In Fortran IV ist diese Handhabung zwar durch Vereinbarungen von der Art der in Algol üblichen überholt, um jedoch soweit wie möglich auch Programme verwenden zu können, die in früheren Versionen verfaßt waren, wird sie nach wie vor beibehalten. Interessanterweise hat dies zu einer neuen Konzeption einer wesentlich weiterreichenden Anwendung geführt (zumindest war es das erste Beispiel für eine derartige Konzeption). Wir meinen damit die Bereitstellung von globalen Konventionen, die eine „Unterlassungsinterpretation" gewährleisten, sobald irgendeine Anweisung mit deklarativem Charakter vergessen worden war. Der große Vorteil einer solchen Konvention besteht darin, daß ein potentieller Anwender einer Sprache von einem großen Teil der Lernarbeit entlastet wird, die er zu leisten hätte, bevor er ein brauchbares Programm schreiben kann.

Wir wollen auch die Frage der Indizierung betrachten. Im Mercury-Autocode ist die Syntax der Variablen folgendermaßen aufgebaut: Wir definieren zunächst die nachstehenden repräsentativen Symbole

⟨*a*-Buchstabe⟩ ::= *a*|*b*|*c*|*d*|*e*|*f*|*g*|*h*|*u*|*v*|*w*|*x*|*y*|*z*
⟨*i*-Buchstabe⟩ ::= *i*|*j*|*k*|*l*|*m*|*n*|*o*|*p*|*q*|*r*|*s*|*t*
⟨π-Buchstabe⟩ ::= ⟨*a*-Buchstabe⟩|π
⟨Vorzeichen⟩ ::= +|−

die zusätzlichen Strukturen für Variable lassen sich nunmehr in der folgenden Form darstellen:

⟨ganzzahlige Variable⟩ ::= ⟨*i*-Buchstabe⟩
⟨reelle Variable⟩ ::= ⟨einfache Variable⟩|⟨indizierte Variable⟩
⟨einfache Variable⟩ ::= ⟨*a*-Buchstabe⟩|⟨*a*-Buchstabe⟩'
⟨indizierte Variable⟩ ::= ⟨π-Buchstabe⟩ ⟨Index⟩
⟨Index⟩ ::= ⟨ganze Zahl⟩|⟨*i*-Buchstabe⟩|
(⟨*i*-Buchstabe⟩ ⟨Vorzeichen⟩ ⟨ganze Zahl⟩)

Somit sind 3, *i*, $(j + 2)$, $(k - 3)$ sämtlich als Indizes zugelassen, jedoch gilt das nicht für $(i + j)$; dieser müßte vorher berechnet werden, etwa durch $m = i + j$. Und während *ab* das Produkt von *a* und *b* bedeutet, wird *ai* interpretiert als a_i (man muß *ia* schreiben, um das Produkt zu erhalten). In Fortran muß, da Namen aus mehreren Buchstaben zugelassen sind, das Multiplikationszeichen explizit angegeben werden (unter dem Einfluß der Lochkartentechnik wurde hierfür der Stern gewählt); Indizes müssen in Klammern gesetzt werden. Die

kuriose Unterscheidung zwischen a-Buchstaben (a-letters) und π-Buchstaben im Autocode „hängt" nach Brooker und Morris (1962) „mit der Anordnung des Materials in dem durch sehr kurze Zugriffszeit ausgezeichneten Speicher des Mercury zusammen. Gegenwärtig müssen wir diese Tatsache als eine der vielen lästigen Eigenschaften akzeptieren, die leicht in praktischen Autocodes auftreten". Es wäre wohl richtiger, zu sagen, daß es sich hierbei um eine Erscheinung handelte, die in allen Sprachen der letzten fünfziger Jahre auftrat, die aber gegenwärtig nicht mehr geduldet werden kann.

Wir werden auch erkennen müssen, daß nur einfache Indizes möglich sind, d.h., das Feld $a(1, 1), \ldots, a(3, 3)$ muß als der Vektor $a(1), \ldots, a(9)$ behandelt werden, während auf $a(r, s)$ nur durch die vorangehende Berechnung $m = 3(r - 1) + s$ Bezug genommen werden kann. In Fortran II waren im Gegensatz hierzu entweder ein oder zwei Indizes erlaubt, und Fortran IV ließ sogar drei Indizes zu. Indizes von Indizes blieben auch weiterhin der Zukunft vorbehalten.

9.5. Vergleichende Betrachtungen zwischen Autocode und Fortran

Sowohl in Autocode als auch in Fortran waren bedingte Sprünge, jedoch nicht bedingte Ausdrücke zugelassen. Die Formate waren jedoch recht verschieden. Im Autocode wurde das einfache „JUMP ⟨label⟩" (d.h. „SPRUNG ⟨Marke⟩") erweitert zu JUMP ⟨Marke⟩, ⟨arithmetischer Ausdruck⟩ ⟨Relationsoperator⟩ ⟨arithmetischer Ausdruck⟩.
In Fortran wurde die unbedingte Sprunganweisung „GO TO ⟨Marke⟩" ersetzt durch
IF (⟨arithmetischer Ausdruck⟩), ⟨Marke⟩, ⟨Marke⟩, ⟨Marke⟩, wobei alle drei Marken den Charakter einer Vorschrift hatten, und es wurde jeweils das erste, zweite oder dritte ausgewählt je nachdem, ob der arithmetische Ausdruck kleiner, gleich oder größer als Null war. Fortran IV enthält Anweisungen folgender Form:

IF (⟨boolscher Ausdruck⟩) ⟨Anweisung⟩.

Beide Sprachen benutzen gelegentlich auch ganzzahlige Marken.

Beide Sprachen wiederum gewährleisteten eine komprimierte Syntax für die Darstellung von Zyklen. Im Autocode konnte die einfache Zuweisung eines Wertes zu einem i-Buchstaben (i-letter) zu der Form

⟨i-Buchstabe⟩ = ⟨ganzzahliger Wert⟩ (⟨ganzzahliger Wert⟩)
⟨ganzzahliger Wert⟩

erweitert und weiter unten durch die Anweisung „RETURN" (Rückkehr) ergänzt werden. Dem i-Buchstaben wurde dann der erste der ganzzahligen Werte zugewiesen; das Programm wurde sodann bis zum RETURN ausgeführt. Danach wiederholte das Programm diesen Block von Instruktionen so oft wie notwendig, wobei der i-Buchstabe jedesmal um den zweiten ganzzahligen Wert erhöht wurde, bis schließlich der Block mit dem Wert des i-Buchstabens ausgeführt wurde, der dem letzten der drei ganzzahligen Werte gleich war; dann wurde das RETURN ignoriert und das Programm fortgesetzt. In Fortran lautete die entsprechende Anweisung

DO ⟨Marke⟩ ⟨ganzzahlige Variable⟩ = ⟨ganzzahliger Ausdruck⟩, ⟨ganzzahliger Ausdruck⟩, ⟨ganzzahliger Ausdruck⟩

mit der Schrittweite als *drittes* Element auf der rechten Seite, wobei die Marke die letzte Instruktion vor dem Rückkehrpunkt identifiziert, der etwa durch die Scheinanweisung „CONTINUE" (fortsetzen) dargestellt werden könnte, falls es unzweckmäßig erscheint, eine auszuführende Instruktion hierzu mit einer Marke zu versehen. Der i-Buchstabe in einer solchen syntaktischen Konstruktion ist unter dem Namen *Laufvariable* bekannt. Die genaue in Anwendung kommende Syntax ist verhältnismäßig uninteressant, die Semantik jedoch enthält allerhand Klippen. Was geschieht z.B., wenn die Laufvariable innerhalb des zu wiederholenden Blocks verändert wird – wird der geänderte Wert bei Hinzunahme der nächsten Schrittweite akzeptiert, annulliert, oder ist ein derartiges Manöver überhaupt unzulässig? In welcher Weise wird der abschließende Test durchgeführt? Im Autocode konnte die zweite Variable positiv oder negativ sein, der abschließende Test jedoch wurde auf Gleichheit vorgenommen, und eine Schleife wie etwa $i = 1\,(2)\,6$ würde immer wieder mit den Werten $i = 1, 3, 5, 7, \ldots$ durchgeführt. In Fortran wurde auf „größer als" getestet, wodurch diese Klippe umgangen wurde, doch mußten alle drei Variablen positiv sein. Welchen Wert hat die Laufvariable beim Verlassen des Zyklus – den letzten akzeptierten oder den ersten zurückgewiesenen? Fragen dieser Art könnten viel Raum beanspruchen und haben heute nur noch historisches Interesse. Wichtig ist, daß ihre Bedeutung heute erkannt worden ist, und die Beantwortung solcher Fragen bildet einen Teil der Spezifikation einer jeden heute vorzuschlagenden Sprache, während früher die Beantwortung solcher Fragen der Konstruktion zumeist des ersten Compilers für diese Sprache überlassen wurde.

Beide Sprachen sehen eine natürliche Behandlung von Unterprogrammen und Funktionen vor, wobei die Parameter, wenn es nur

irgend möglich ist, bei der Berechnung bis auf die Stufe einer Maschinenadresse oder auf einen einzigen Wert im Falle eines Ausdrucks abgebaut werden, bevor in das Unterprogramm eingesprungen wird. Es handelt sich hier also in Wirklichkeit um den Sachverhalt, den man seit dieser Zeit als Aufruf durch einen einfachen Namen bezeichnet hat.

Sobald indizierte Variable mit einbegriffen werden, muß der Maschine ein gewisser Hinweis über den erforderlichen Speicherbedarf gegeben werden, in dem man die Anzahl der Komponenten spezifiziert, die jede Variable durchlaufen kann. Im Autocode geschah das durch Hinweise der Form $a \rightarrow 20$, wobei vorausgesetzt wurde, daß zusätzlich zu den einfachen Variablen a und a' Speicherraum für einen Vektor a, dessen Indizes von 0 bis 20 liefen, benötigt wurde. Eine Folge solcher Hinweise veranlaßte eine fortlaufende Zuweisung von Speicherplätzen. Wäre z.B. auf unser $a \rightarrow 20$ der Hinweis $c \rightarrow 5$ gefolgt, so würde ein Aufruf von a_{22} auf c_1 führen. Ein solcher Aufruf wäre zwar normalerweise das Ergebnis eines Programmierungsfehlers, doch wurden solche Möglichkeiten auch gelegentlich ausgenutzt. Das entsprechende Programmierungselement in Fortran war die „Dimensionsanweisung" (DIMENSIONstatement), die etwa in der Form

DIMENSION A (2, 3)

besagte, daß A den Namen *nicht einer einfachen Variablen* (im Gegensatz zu Autocode) *oder einer Funktion darstellte, sondern ein Feld kennzeichnete*, dessen Indizes die Werte von 1 bis 2 und die Werte von 1 bis 3 durchliefen. Ein Überschreiten der angegebenen Grenzen wurde signalisiert.

Schließlich mußten wegen des geringen Umfanges des Hauptspeichers des Mercury die Autocodeprogramme in „Kapitel" unterteilt werden, wobei jedes so zu bemessen war, daß es im Hauptspeicher untergebracht werden konnte (falls ein Kapitel diese Grenzen überschritt, wurde auf ein Signal des Monitorsystems hin das Programm an den Programmierer zur Korrektur zurückgegeben). Jedes Kapitel hatte seine eigenen Marken, und es waren spezielle Anweisungen, „across" oder „down", für den Übergang in andere Kapitel vorhanden („down" implizierte einen Rücksprung „up" zu einem späteren Zeitpunkt). Das Kapitel 0 war stets das letzte Kapitel des Programms und bildete das Signal dafür, daß die Ausführungsphase in Angriff zu nehmen war. Ein Fortran-Programm wies in seiner Organisation durch Namen bezeichnete Unterprogramme und ein Haupt-

programm ohne Namensbezeichnung auf. Durch „CALL ⟨Unterprogramm⟩" (Aufruf) wurde ein Absprung in ein Unterprogramm bewirkt und durch „RETURN" der Rücksprung ins aufrufende Programm. Es war stets ein Argument der Fortran-Anhänger, daß kein Programm mehr als einmal übersetzt werden mußte, denn ein vollständiges Programm konnte aus einzelnen in Fortran geschriebenen Teilen und bereits in Maschinencode compilierten Teilen zusammengesetzt werden. Diese Maßnahme wurde getroffen, weil die Compilierung mitunter recht langsam vor sich gehen konnte. Durch eine Vereinbarung, die mit „COMMON" (gemeinsam) eingeleitet wurde, wurde gekennzeichnet, daß eine Variable nicht nur eine lokale Bedeutung für ein Unterprogramm hatte, sondern eine globale Geltung besaß. Zu dieser Konzeption wurde noch eine durch das Wort „EQUIVALENT" (gleichwertig) eingeleitete Vereinbarung hinzugefügt, die es gestattete, daß manche Speicherbereiche in verschiedenen Routinen durch verschiedene Namen aufgerufen werden konnten. Unglücklicherweise haben verschiedenartige Interpretationen des Zusammenspiels zwischen diesen beiden Vereinbarungsarten zu einer Inkompatibilität zwischen Fortran II und Fortran IV geführt, so daß Programme des Fortran II nicht immer durch einen Fortran IV-Compiler übersetzt werden können, obwohl es sich bei der ersten um eine Teilsprache der zweiten handelt.

Im Ergebnis dieser Schwierigkeit, ferner veranlaßt durch eine Anzahl lokaler Spielarten, wurde 1962 eine Kommission eingesetzt, so daß im Jahre 1964 der Entwurf einer Standardspezifikation für zwei Sprachen Fortran und Basic Fortran veröffentlicht wurde (Heising, 1965). Diese beiden Sprachen sind soweit wie nur irgend möglich mit den verschiedenen existierenden Versionen von Fortran IV bzw. Fortran II kompatibel, und Basic Fortran ist als eine echte Untermenge von Fortran definiert. Diese Fortran-Spezifikation ist ein Dokument, das etwa 16000 Worte enthält, während das Dokument für Basic Fortran eine gekürzte Kopie mit einigen sich aus der Kürzung ergebenden Modifikationen darstellt (die Anzahl der Worte des vorliegenden Buches liegt bei etwa 60000).

9.6. Jovial

Im Jahre 1958 haben die Kommissionen, die an der Erarbeitung des Algol beteiligt waren, einen Interimbericht veröffentlicht, der, wie sich nachträglich herausstellte, zwei Hauptergebnisse gezeitigt hat.