atp-Softwaretechnik

Tabelle
Durchf
Verschi

Die Programmiersprache CHILL

J. F. H. Winkler, München

Die Programmiersprache CHILL wurde in der Zeit von 1975 bis 1983 vom CCITT zum Zwecke der Programmierung von Vermittlungsrechnern entwickelt. Vermittlungsrechner sind Prozeßrechner, und daher enthält CHILL auch Sprachelemente, die für die Programmierung von Prozeßrechnern wichtig sind. Beispiele hierfür sind nebenläufige Prozesse und Darstellungsspezifikationen für Daten. Die Programme von Vermittlungssystemen sind außerdem sehr umfangreich und komplex. Daher enthält CHILL auch Sprachelemente für das Programmieren im Großen, und zwar ein sehr flexibles Modulkonzept.

In der vorliegenden Arbeit wird CHILL in Anlehnung an den Vorschlag [1] dargestellt.

Zeile
1
2
3
4
5
6
7
8
9

[1]) W

1. Geschichte von CHILL

CHILL (CCITT High Level Language) wurde in der Zeit von 1975 bis 1983 vom CCITT (Comité Consultatif International Télégraphique et Téléphonique), einem Ausschuß der ITU (International Telecommunications Union), zum Zwecke der Programmierung von Vermittlungsrechnern entwickelt. Im CCITT arbeiten die Fernmeldeverwaltungen der meisten Staaten weltweit zusammen, um Richtlinien und Normen für die Telekommunikation zu definieren. CHILL ist als Empfehlung Z. 200 (1984) vom CCITT im Herbst 1984 anläßlich der Hauptversammlung in Torremolinos verabschiedet worden. Eine andere Empfehlung des CCITT ist die bekannte V.24-Schnittstelle.

Die Bedeutung des Rechnereinsatzes in der Vermittlungstechnik wurde beim CCITT bereits in den 60er Jahren gesehen und führte in der Hauptversammlung im Oktober 1968 in Mar del Plata, Argentinien, zur Formulierung der Studienfrage 7/XI („Study of methods for the specification of program logic for stored program controlled telephone exchanges") [2]. Die Arbeiten in der Studienperiode 1968/72 führten zu keinen konkreten Ergebnissen. Daher wurde für die nächste Studienperiode 1973/76 die genauere Studienfrage 8/XI („High level programming language for SPC-telephone exchanges") gestellt [3]. Ähnlich wie bei Ada[1]) wurden in einer ersten Phase existierende Programmiersprachen auf ihre Eignung für den vorgesehenen Zweck untersucht. Keine der 27 untersuchten Sprachen erfüllte alle Anforderungen der Beteiligten, so daß im Jahre 1975 eine Spezialistengruppe mit Vertretern aus sieben Verwaltungen und Firmen gegründet wurde. In dieser Gruppe waren vertreten ITT, L. M. Ericsson, Nordic PTT, NTT Japan, Philips, Siemens und UKPO. Ein erster Sprachentwurf lag 1977 in Form des „Blue Documents" vor [4]. Erfahrungen bei der Erstellung der ersten Übersetzer führten zu einer Überarbeitung von [4]. Die erste offizielle Version von CHILL wurde 1980 als Empfehlung Z.200 [5] verabschiedet. In der Studienperiode 1981/84 wurden eine Reihe von weiteren Elementen und Verbesserungen in die Sprache eingebracht und die jetzt gültige Definition im Herbst 1984 als Z.200 (1984) [6] verabschiedet.

Entsprechend den Anforderungen des Anwendungsgebietes, welches durch hohe Verfügbarkeit der Systeme (z. B. ≥ 0.9999) und große Komplexität der Programme gekennzeichnet ist, waren die wesentlichen Entwicklungsziele von CHILL:

- zuverlässige Programme,
- effiziente Objektprogramme,
- modulare und gut strukturierte Programme,
- Maschinenunabhängigkeit,
- leicht zu erlernen und zu benutzen.

Die nun vorliegende Sprache CHILL ist eine Programmiersprache der Algol-Tradition, zu welcher z. B. auch Algol 60, Algol 68, Pascal und Ada gehören. CHILL gehört zur Gruppe der modernen, umgangreichen Programmiersprachen. Andere Vertreter dieser Gruppe sind Ada [7], Mesa [8] und Smalltalk [9]. Diese Sprachen sind dadurch charakterisiert, daß sie sowohl Sprachelemente für das Programmieren im Kleinen als auch für das im Großen enthalten. Außerdem enthalten diese Sprachen ein reichhaltiges Repertoire von Sprachelementen für die professionelle Systemprogrammierung. CHILL ist daher keine „Telefonsprache", denn die Grundoperationen der Vermittlungstechnik sind auf einer viel höheren Abstraktionsebene angesiedelt als die in heutigen Programmiersprachen üblichen Operationen.

2. Allgemeine Bewertungskriterien

a) Allgemeine Orientierung

CHILL steht in der Tradition der blockstrukturierten, algorithmischen Programmiersprachen. In der Sprache selbst sind die Einflüsse von Pascal, Concurrent Pascal, PL/1 und Modula-2 festzustellen.

[1]) Ada ist ein eingetragenes Warenzeichen des AJPO (Ada Joint Program Office).

Die wesentlichen Gruppen von Sprachmerkmalen sind:

Module und Monitore (Region)

Sie sind Einheiten der getrennten Übersetzung mit der Möglichkeit, die Schnittstelle als Spezifikationsmodul auszubilden. Module können auch geschachtelt sein, was besonders für große Systeme wesentlich ist. Das Programm des EWSD [10] besteht z. B. aus einigen Tausend Modulen. Ähnlich wie in Ada durch die Untereinheiten (subunits) können die Rümpfe eingeschachtelter Module textuell ausgelagert werden, um die Programmbausteine auf eine gut handhabbare Größe beschränken zu können.

Der Ex- und Import von Größen kann erfolgen, spezifisch für einzelne Größen, für Gruppen von Größen und für alle Größen eines Bausteins. Beim Ex- und Import sind Umbenennungen möglich, was insbesondere dann von großer Bedeutung ist, wenn bereits vorhandene Bausteine eingesetzt werden sollen, bei deren Erstellung kein Einfluß auf die Namensvergabe bestand.

Nebenläufige (parallele) Prozesse

Wie für Sprachen zur Programmierung von Prozeßrechnern üblich [11], enthält auch CHILL das Konzept der nebenläufigen Prozesse. Die Prozesse sind im wesentlichen Prozeduren, die nebenläufig zueinander ausgeführt werden können. Zur Prozeßkoordinierung (Sychronisation und Kommunikation) stehen Monitore [12; 13] sowie gepufferte und ungepufferte asynchrone Kommunikation zur Verfügung.

Ausnahmebehandlung

Bei der Ausnahmebehandlung bietet CHILL Sprachelemente, die im wesentlichen den Möglichkeiten von Ada und COBOL entsprechen.

Datentypen

An Datentypen enthält CHILL die in modernen Programmiersprachen üblicherweise vorhandenen: Zahlen, Aufzählungstypen, Verbunde (Records), Reihungen (Arrays), Verweise, Dateien und Mengen skalarer Größen. Für Verbunde und Reihungen ist es möglich, die Abbildung auf den Speicher im Programm zu spezifizieren (representation specifications). Dadurch sind insbesondere Bitmanipulationen, wie sie bei der Programmierung von Prozeßrechnern häufig vorkommen, leicht und in problembezogener Weise möglich. Werte von strukturierten Typen können als Aggregate notiert werden, so daß das etwas mühsame komponentenweise Ansprechen, wie z. B. in Pascal, entfällt. Prozedurtypen können verwendet werden, um Prozedurvariable und Prozeduren als Parameter zu realisieren.

Sequentielle Ablaufsteuerung

Die Anweisungen für die sequentielle Ablaufsteuerung umfassen die Sequenz, Fallunterscheidungen (IF und CASE), Schleifen, Unterprogrammaufrufe und Sprunganweisungen (GOTO, EXIT, RETURN). Die CASE-Anweisung ist in allgemeinerer Form als üblich realisiert: es kann eine Folge von Auswahlausdrücken angegeben werden. Die Alternativen sind dann durch entsprechende Tupel von Werten markiert. Damit lassen sich sehr leicht Entscheidungstabellen formulieren, welche die Lesbarkeit eines Programmes sehr verbessern können.

Ein- und Ausgabe

Es können Dateien von beliebigem Typ definiert und gezielt externen Datenobjekten zugeordnet werden. Der Zugriff auf die Dateikomponenten (Sätze) ist sequentiell und wahlfrei möglich. Text-Ein- und Ausgabe ist derzeit noch nicht enthalten und soll in der laufenden Studienperiode in die Sprache aufgenommen werden.

b) Eignung für gute Programmierung

CHILL entspricht dem Stand der Technik im Bereich der algorithmischen Programmiersprachen. Gute Programmierung wird unterstützt durch die Sprachmittel zum Programmieren im Großen: Module, getrennte Übersetzung, Modulschachtelung, Untereinheiten und vielseitige Ex- und Importkonstruktionen. Im Bereich der Nebenläufigkeit wird durch die Monitore eine gute Strukturierung erreicht, die Vorteile gegenüber der Verwendung von Semaphoren aufweist. Für den Datenaustausch zwischen Prozessen stehen Mittel zur asynchronen Kommunikation zur Verfügung, die eine direkte Formulierung des Nachrichtenaustausches zwischen Prozessen ermöglichen. Für das Programmieren im Kleinen stehen die üblichen Grundkonstruktionen der strukturierten Programmierung zur Verfügung, wobei die CASE-Anweisung deutlich besser ist als in den meisten anderen Sprachen.

Die strenge Typbindung, die nur explizit umgangen werden kann, trägt ebenfalls dazu bei, zuverlässige Programme zu formulieren. Außerdem gestattet sie, viele Fehler bereits durch den Übersetzer feststellen zu lassen.

Die Ausnahmebehandlung führt vor allem zu kürzeren Programmen. Erhöht werden durch sie die Übersichtlichkeit, Lesbarkeit, Funktionalität, Robustheit und Zuverlässigkeit der Programme.

Was in CHILL, wie auch in den anderen algorithmischen Sprachen fehlt, ist die Möglichkeit, auch die Semantik von Schnittstellen abstrakt zu beschreiben, also nicht nur die rein syntaktische Schnittstelle eines Unterprogrammes mittels der UP-Definition zu spezifizieren, sondern auch die Wirkung dieses Unterprogrammes in dieser Spezifikation zum Ausdruck zu bringen [14].

c) Definition der Sprache

CHILL ist definiert in der Empfehlung Z.200 (1984) des CCITT [6]. Das Dokument ist in einem halbformalen Stil in natürlicher Sprache (Englisch) abgefaßt und bezüglich der Formalität zwischen den Dokumenten von z. B. Ada [7], Pascal [15] einerseits und der Definition von PL/I [16] andererseits einzuordnen. Die Definition einer Sprachkonstruktion ist jeweils gegliedert in die Abschnit-

te Syntax, Semantik, statische Eigenschaften, dynamische Eigenschaften, statische Bedingungen und dynamische Bedingungen. Semantische Beziehungen, wie z. B. die Vorschriften über Typverträglichkeiten, sind zentral in einem Kapitel in sehr präziser Art und Weise definiert. Eine formale Definition mit Hilfe von VDM (Vienna Definition Method [17]) wurde ebenfalls als CCITT-Dokument angenommen [18].

d) Verfügbarkeit und Bewährtheit in der Praxis

Der Einsatz von CHILL beschränkt sich bisher hauptsächlich auf den Telekommunikationsbereich, in welchem die Sprache auch entstanden ist. Dieser Bereich gehört sicher auch zur Automatisierungstechnik im weiteren Sinne, beide Bereiche scheinen heute aber noch relativ isoliert voneinander zu sein. Erste Anwendungen außerhalb des Telekommunikationsbereiches werden in neuerer Zeit ebenfalls bekannt [19]. Innerhalb seines aktuellen Anwendungsgebietes ist CHILL bei den hauptsächlichen Anbietern von Telekommunikationssystemen implementiert worden und auf unterschiedlichen Rechnern verfügbar. Wegen der Größe der Systeme ist es im Telekommunikationsbereich zur Zeit üblich, die Programme auf einem Entwicklungsrechner zu entwickeln und dann die erzeugten Objektprogramme in den Vermittlungsrechner zu laden.

Eine Übersicht aus dem Jahre 1984 [20] erwähnt 10 bis 25 verschiedene Übersetzer. Die genaue Zahl hängt davon ab, ob man einen Übersetzer mit mehreren Codegeneratoren mehrfach zählt oder nicht. Die Übersetzer laufen in der Regel auf großen und mittleren Rechnern, während als Zielmaschinen alle Rechnerarten vorkommen.

Bei den Herstellern wird CHILL nicht nur zur Programmierung der Vermittlungsprogramme verwendet, sondern auch für die zur Programmierung erforderlichen Werkzeuge. Bei Siemens z. B. sind die CHILL-Übersetzer und die weiteren Werkzeuge der CHILL-Entwicklungsumgebung in CHILL programmiert. Da verschiedene Hersteller bereits in der Zeit vor der Fertigstellung der Sprachdefinition 1980 mit der Implementierung begannen, ist meistens nicht der 1984 definierte Sprachumfang implementiert. Darauf wird unter dem Punkt „Übertragbarkeit" weiter eingegangen.

CHILL hat sich in der Praxis bereits bewährt, denn es sind weltweit schon eine ganze Reihe von in CHILL programmierten Vermittlungssystemen im Einsatz. Bei der Deutschen Bundespost ist die Situation im Bereich der öffentlichen Fernsprechnetze so, daß neue Vermittlungssysteme als digitale Systeme installiert werden. Derzeit sind dies die Systeme „System 12" von SEL/ITT und EWSD (Elektronisches Wählsystem Digital) von Siemens. EWSD besteht aus mehreren Millionen Zeilen Code, gegliedert in einige Tausend Module. Das System ist bisher in mehr als 1100 Exemplaren mit insgesamt 5,2 Millionen Anschlußeinheiten in 21 Ländern verkauft worden. Darüberhinaus wird CHILL auch für andere Vermittlungssysteme, z. B. im Nebenstellenbereich, verwendet.

e) Übertragbarkeit

Die höheren Programmiersprachen sind im Prinzip maschinenunabhängig konzipiert, und daher ist auch Übertragbarkeit im Prinzip möglich. Wie gut die Übertragbarkeit in einem speziellen Fall dann wirklich ist, hängt von zwei Faktoren ab.

Umfang der Sprache

Bei einer Sprache mit nur wenigen Sprachelementen (wie z. B. Standard-Pascal) wird es bei industriellen Anwendungen immer wieder vorkommen, daß Funktionen, welche in der Sprache nicht enthalten sind, durch Betriebssystemaufrufe oder hinzugebundene Assemblerprogramme realisiert werden müssen [21]. Da solche Erweiterungen in der Regel dann bei verschiedenen Herstellern verschieden ausfallen, wird durch einen geringen Funktionsumfang der Sprache selbst die Übertragbarkeit (besser wäre eigentlich das Übertragbarkeitspotential) beeinträchtigt [22]. Da CHILL einen großen Funktionsumfang hat, ist sein Übertragbarkeitspotential hoch.

Faktische Implementierungssituation

Ein großes Übertragbarkeitspotential allein garantiert noch keinen hohen Übertragbarkeitsgrad. Wenn die Sprache nicht einheitlich auf vielen verschiedenen Rechnern implementiert ist, dann ist die faktische Übertragbarkeit eingeschränkt. Im derzeitigen Anwendungsbereich von CHILL wird Übertragbarkeit hauptsächlich zwischen verschiedenen Rechnern eines Herstellers verlangt. Ähnlich wie bei anderen großen Programmen in der Automatisierungstechnik ist es bei den großen Vermittlungssystemen nicht üblich, sie von der Rechenanlage eines Herstellers auf die eines anderen zu übertragen. Der Anwender eines Vermittlungssystems kauft in der Regel ein komplettes System bestehend aus Apparatur und Programmausrüstung.

Die Lage bei CHILL ist daher ähnlich der bei PEARL: Übertragbarkeit zwischen den Implementierungen verschiedener Hersteller ist gegeben, wenn man sich auf eine Teilmenge der vollen Sprache beschränkt [23]. Zwischen verschiedenen Rechnern eines Herstellers ist eine Übertragbarkeit in der Regel gegeben.

f) Unterstützung

Beim CCITT besteht weiterhin eine Arbeitsgruppe, die sich mit der Pflege, Fortentwicklung und Förderung der Verbreitung der Sprache beschäftigt. In etwa zweijährigem Rhythmus findet eine internationale CHILL-Tagung statt, bei welcher die Sprache und ihre Anwendungen behandelt werden. Neben den in der Einführung genannten Sprachdokumenten gibt es beim CCITT eine „Introduction to CHILL" [24] und ein „CHILL User Manual" [25; 26].

Darüber hinaus gibt es weitere Bücher über die Sprache [27; 28; 29]. Eine CHILL-Bibliografie wurde in [30] veröffentlicht. Die Ausbildung von CHILL-Programmierern erfolgt bisher hauptsächlich bei den Systemherstellern

selbst, die dazu entsprechende Kurse entwickelt haben. Einer Übersicht von 1982 nach wurden bis dahin weltweit mehr als 5000 Programmierer in CHILL ausgebildet (CCITT TD 363-E, June 1982). Eine informelle Zeitschrift, das CHILL-Bulletin, unterrichtet einen weltweiten Leserkreis über CHILL betreffende Fragen und Entwicklungen.

g) Zertifizierung

Um die Übereinstimmung von Übersetzern mit der Sprachdefinition zu überprüfen, wurden zwei Folgen von Testprogrammen für CHILL entwickelt. Diese wurden von verschiedenen Herstellern zum Test ihrer Übersetzer verwendet. Ähnlich wie bei PEARL gibt es bisher aber keine offizielle Zertifizierung von Übersetzern durch das CCITT oder eine vergleichbare Stelle.

3. Technische Bewertungskriterien

Bei der folgenden Darstellung der technischen Eigenschaften von CHILL wird die Kenntnis der wesentlichen Sprachelemente von Pascal und PEARL vorausgesetzt.

a) Sicherheit

Wie in anderen modernen Programmiersprachen auch, sorgen viele Sprachelemente von CHILL dafür, daß Programmierfehler möglichst vor dem Ablauf oder beim Ablauf festgestellt werden können.

Der Zugriff zu Datenobjekten läßt sich durch das in der Sprache vorhandene Modulkonzept wirksam kontrollieren. In CHILL können verschiedene Formen der Datenabstraktion realisiert werden [14]. Das Monitorkonzept erlaubt darüberhinaus die zeitliche Koordination des Zugriffs nebenläufiger Prozesse zu gemeinsam benutzten Daten. Bei der Prozeßkommunikation über die sogenannten SIGNALs ist es möglich, Nachrichten genau an eine bestimmte Prozeßinkarnation zu senden. Außerdem kann angegeben werden, daß nur Prozesse eines bestimmten Typs als Empfänger eines bestimmten SIGNAL auftreten können. Innerhalb geschachtelter Unterprogramme und Blöcke gilt die bekannte „globale Sicht".

Beim Programmieren im Kleinen sorgt die strenge Typbindung dafür, daß die Kompatibilität zwischen Operanden und Operationen durch den Übersetzer überprüft werden kann. Dabei zählen zu den Operationen z. B. auch die Zuweisung, Unterprogrammaufrufe und Indizierungen oder Selektionen. Indexwerte und Auswahlausdrücke für die Auswahl von Verbundvarianten werden, falls erforderlich, beim Programmablauf auf Zulässigkeit überpüft. Durch die Möglichkeit, abgeleitete Typen (NEWMODE in CHILL) zu definieren, lassen sich auch Typen unterscheiden, die den gleichen Wertebereich haben oder sonst sehr ähnlich sind. Damit kann man z. B. Variable für Temperatur und für Länge auseinanderhalten, auch wenn sie den gleichen Wertebereich haben:

```
NEWMODE Temperatur = RANGE(0:500),
        Laenge     = RANGE(0:500);
DCL TempVar Temperatur INIT := 10,
    LaengVar Laenge    INIT := 20;

TempVar := LaengVar; /* ist nicht erlaubt */
```

Dadurch können unzulässige Verknüpfungen solcher Variablen durch den Übersetzer festgestellt werden. Durch die Initialisierung von Variablen in der Vereinbarung kann prinzipiell die unsichere Situation vermieden werden, in welcher der Wert einer Variablen undefiniert ist. Auch dies trägt zur Sicherheit der Programme bei.

b) Zuverlässigkeit

Der Anwendungsbereich von CHILL zeichnet sich durch hohe Zuverlässigkeitsanforderungen aus. Zum Beispiel läßt die Deutsche Bundespost bei digitalen Ortsvermittlungsstellen eine Ausfallzeit von höchstens 2 Stunden pro Jahr und bei Fernvermittlungsstellen von höchstens 1 Stunde pro Jahr zu. Spracheigenschaften, die zur Zuverlässigkeit und damit auch zur Verfügbarkeit beitragen, wurden bereits unter Abschnitt 3a angegeben.

Die Zuverlässigkeit von Programmen wird außerdem durch die in CHILL vorhandene Ausnahmebehandlung erhöht. Dabei kann eine problemspezifische Reaktion auf Laufzeitfehler, wie z. B. Division durch Null oder unzulässiger Indexwert, formuliert werden. Vordefinierte Ausnahmen gibt es für alle Sprachelemente, bei denen Laufzeitfehler auftreten können.

Bei der Konstruktion möglichst korrekter Programme wird der Programmierer in CHILL außerdem durch die ASSERT-Anweisung unterstützt. Die Anweisung

ASSERT ⟨logischer Ausdruck⟩;

bewirkt, daß beim Durchlaufen überprüft wird, ob der logische Ausdruck wahr ist. Ist dies nicht der Fall, dann wird die Ausnahme ASSERTFAIL ausgelöst. In einfachen Fällen kann bereits der Übersetzer durch eine statische Programmanalyse entscheiden, ob die Bedingung jemals verletzt werden kann oder nicht.

c) Lesbarkeit

Die gute Lesbarkeit von CHILL-Programmen wird durch folgende Eigenschaften erreicht:

- *„Beliebig" lange Bezeichner:* Jede Implementierung hat natürlich eine zulässige Maximallänge, aber diese kann z. B. auch gleich der maximalen Zeilenlänge sein.
- *Strukturierte Anweisungen* werden im Gegensatz z. B. zu Pascal mit einem Schlüsselwort abgeschlossen. Im Gegensatz zu Modula-2 werden für verschiedene Anweisungen verschiedene abschließende Schlüsselworte verwendet; eine IF-Anweisung endet mit FI, eine CASE-Anweisung mit ESAC und eine Schleife mit OD. Programmbausteine wie Unterprogramme oder Module enden einheitlich mit „END ⟨Bezeichner⟩;", wobei der ⟨Bezeichner⟩ optional ist. Falls er vorhanden ist, muß er der Name des betreffenden Bausteins sein.

- Die Gliederung des Programmes in *Module* und *Monitore*.
- Die *mehrdimensionale CASE-Anweisung:* Dafür wird im Abschnitt „Sequentielle Ablaufstrukturen" ein Beispiel gegeben.
- *Benannte Anweisungen:* Alle Anweisungen können durch einen vorangestellten Namen benannt werden, und dieser Name kann am Ende wiederholt werden. Wenn er wiederholt wird, dann muß er auch mit dem vorangestellten übereinstimmen. Ein Beispiel dafür wird im Abschnitt „Sequentielle Ablaufstrukturen" gegeben.
- *Aufzählungstypen:* Dadurch können insbesondere Zustände im technischen Prozeß mnemotechnisch gut benannt werden.
- *Aggregate* für Werte von Reihungen (Arrays) und Verbunden (Records). Dies ermöglicht, wie in Ada, die aus der Mathematik her gewohnte Schreibweise für Mengen, Tupel, Vektoren und Matrizen im Programm zu verwenden. Wenn X z. B. ein Vektor mit drei Komponenten ist, dann kann man schreiben:

```
X := [2, 4, 8];
```

Über die in der Mathematik übliche Schreibweise hinaus kann man, wie in Ada, Aggregate auch in einer Schlüsselwortschreibweise notieren. Für einen Vektor Y mit 10 Komponenten kann man schreiben:

```
Y := [(1):5, (3,5):7, (7:9):6, ELSE:0];
```

Dadurch erhält Y[1] den Wert 5, Y[3] und Y[5] den Wert 7, Y[7], Y[8] und Y[9] den Wert 6 und Y[2], Y[4], Y[6] und Y[10] den Wert 0. Diese Schreibweise für Aggregate ist auch wichtig für Verbundaggregate. Enthält z. B. ein Verbund V die Komponenten LAENGE, HOEHE, BREITE, dann kann man schreiben:

```
Y := [.LAENGE:20, .HOEHE:10, .BREITE:7];
```

d) Erlernbarkeit

Die Erlernbarkeit der einzelnen Sprachelemente von CHILL entspricht der von Pascal oder anderen höheren Programmiersprachen. Da CHILL aber wesentlich mehr Sprachelemente als Pascal enthält, ist es bezüglich der Erlernbarkeit eher im Bereich von Full-PEARL oder Ada anzusiedeln.

e) Sequentielle Ablaufstrukturen

Die **sequentiellen Ablaufstrukturen** von CHILL sind die folgenden:

- *Sequenz* durch Aneinanderreihung. Wie bei Ada gehört auch in CHILL ein abschließendes Semikolon zu der betreffenden Anweisung hinzu. Eine Zuweisung hat daher folgenden Aufbau:

```
V: = Wert;
```

Dadurch daß das Semikolon zur Anweisung gehört, und nicht als „Sequentialisierungsoperator" gedeutet wird, und durch die geschlossene Form der strukturierten Anweisungen wird das seit Algol 60 bis hin zu Pascal vorhandene Problem des Semikolons vor ELSE und END gelöst. Unabhängig davon, wo die Zuweisung „V := Wert;" steht, wird sie stets genau in dieser Form geschrieben; auch innerhalb einer IF-Anweisung:

```
IF Bedingung
THEN V: = Wert;
ELSE  V: = 0;
FI;
```

- *IF-Anweisung.* Hier ist die Form gewählt worden, bei welcher eine Häufung schließender Wortsymbole vermieden wird:

```
IF      Bedingung-1  THEN Aktion-1
ELSIF   Bedingung-2  THEN Aktion-2
ELSIF   ....
:
:
ELSE Aktion-n   FI;
```

Durch Verwendung von ELSIF genügt auch bei einer solchen Kaskade *ein* abschließendes FI.

- *CASE-Anweisung.* Ihre prinzipielle Struktur ist:

```
CASE AA1, AA2, .. ,AAn OF
     M11, M12,...M1n1: Aktion-1;
     M21, M22,...M2n2: Aktion-2;
        :
        :
     ELSE Aktion-m
ESAC;
```

Dabei sind die AA*i* die (diskreten) Auswahlausdrücke und die M*ij* entsprechende Wertetupel. Ein praktisches Beispiel für diese mehrdimensionale CASE-Anweisung ist der in Bild 1 dargestellte Ausschnitt aus einem Sortierprogramm [6].

An diesem Beispiel erkennt man deutlich den Entscheidungstabellencharakter der mehrdimensionalen CASE-Anweisung. Wie in PL/I und PEARL wird in CHILL der Kommentar durch die Zeichenkombinationen „/*" und „*/" begrenzt.

- *Schleifen.* Schleifen werden in CHILL durch DO und OD begrenzt. Die Steuerung der Schleifenausführung geschieht wie in anderen Sprachen über Steuerungsklauseln, die entweder unmittelbar auf das DO folgen, oder über EXIT-Anweisungen, die im Schleifenrumpf an beliebigen Stellen stehen dürfen. Zählschleifen werden durch die bekannte FOR-Klausel gesteuert:

```
DO FOR I : = 1 TO N;
   Prod : = V1[I] * V2[I];
OD;

DO FOR  I IN Index-Type;
   Prod : = V1[I] * V2 [I];
OD;
```

```
CASE  OUTOFFILE(infiles(FALSE)), OUTOFFILE(infiles(TRUE))  OF
        (TRUE),                  (TRUE)  : EXIT MergeFiles;      /* both files are empty */
        (TRUE),                  (FALSE) : flag: = TRUE;         /* one file is empty */
        (FALSE),                 (TRUE)  : flag: = FALSE;        /* one file is empty */
        (FALSE),                 (FALSE) : flag : = buffers(FALSE).key > buffers(TRUE).key;
ESAC;
```

Bild 1. Beispiel für eine mehrdimensionale CASE-Anweisung (Ausschnitt aus einem Sortierprogramm [6]).

In einer Zählschleife sind auch mehrere Laufbereiche möglich:

```
DO FOR J := 1 TO I-1, I + 1 TO N;
```

Eine Schleife ohne explizite Steuerung kann realisiert werden mit

```
DO FOR EVER;
```

Die Schrittweite einer Zählschleife kann jede ganze Zahl ungleich Null sein:

```
FOR I := 0 BY 10 TO N;
```

Soll die Zählung in absteigender Richtung erfolgen, so wird dies durch das Schlüsselwort DOWN ausgedrückt:

```
FOR I := N BY 10 DOWN TO 0;
```

Alle Ausdrücke in der Schleifensteuerung können allgemein dynamische Ausdrücke von beliebigem diskreten Typ sein.

Eine Iterationsschleife wird durch folgende Steuerung gebildet:

```
DO WHILE Bedingung;
```

wobei wie üblich „Bedingung" ein logischer Ausdruck sein muß.

Zähl- und Iterationssteuerung können auch wie in Algol60 miteinander kombiniert werden:

```
FOR I := 0 TO N WHILE SUM ≤ MAX;
```

Oft wird die Bedingung zur Beendigung einer Schleife innerhalb des Schleifenrumpfes berechnet und nicht am Anfang, wie es bei der Iterationsschleife der Fall ist [31]. Um dies zu ermöglichen, gibt es die Anweisung

```
EXIT Name;
```

wobei „Name" der Name einer strukturierten Anweisung oder eines Moduls ist, und die EXIT-Anweisung innerhalb der genannten Konstruktion stehen muß.

Ein typisches Beispiel für solch eine Schleife ist:

```
SuchSchleife:  DO FOR EVER;
                  Aktion1
                  IF Gefunden
                  THEN EXIT SuchSchleife; FI;
                  Aktion2
               OD SuchSchleife;
```

Die **elementaren Anweisungen** von CHILL im Bereich des sequentiellen Programmablaufes sind:

- *Zuweisung* in der einfachen Form

  ```
  Var := Ausdruck;
  ```

  Außer dieser einfachen Form gibt es auch die Zuweisung mit mehrfacher linker Seite,

  ```
  Var1, Var2, Var3 := Ausdruck;
  ```

  die es bereits in Algol60 gegeben hat.

  Wie in Algol68 können gewisse einfache Zuweisungen in kompakterer Form geschrieben werden, die auch zu effizienterem Objektcode führen kann. Statt

  ```
  AktuelleZeile := AktuelleZeile + 1;
  ```

  kann geschrieben werden

  ```
  AktuelleZeile + := 1;
  ```

- *Prozeduraufruf*

  ```
  CALL P(A, B, C);
  ```

  wobei das Schlüsselwort CALL auch entfallen kann.

- *RESULT-Anweisung*

  ```
  RESULT Ausdruck;
  ```

  Mit dieser Anweisung wird das Ergebnis der Funktion festgelegt, in welcher die Anweisung enthalten ist. Die Funktion wird dabei aber noch nicht beendet. Dies erfolgt erst durch

  ```
  RETURN;
  ```

  Die Wirkung beider Anweisungen kann mit

  ```
  RETURN Ausdruck;
  ```

  erreicht werden.

- *Sprunganweisung*

  ```
  GOTO Marken-Name;
  ```

- *Zusicherung*

  ```
  ASSERT Bedingung;
  ```

  Wenn der Wert von „Bedingung" falsch ist, dann wird die Ausnahme ASSERTFAIL ausgelöst.

- *Ausnahmebehandlung.* Zum Bereich der sequentiellen Ablaufsteuerung gehört weiterhin die Ausnahmebehandlung. Hier hat CHILL ein Konzept, welches der Ausnahmebehandlung in Ada und COBOL ähnelt. Jeder Anweisung und jedem Block kann eine Liste von Ausnahmebehandlern angefügt werden:

```
A := B + C
     ON (OVERFLOW, RANGEFAIL):
          A := Max;
     END;
```

Es gibt vordefinierte Ausnahmen für die durch die Sprachkonstruktionen bedingten Ausnahmesituationen, und außerdem kann der Benutzer weitere problemspezifische Ausnahmen definieren. Mit der CAUSE-Anweisung kann eine Ausnahme explizit ausgelöst werden:

```
CAUSE OVERFLOW;
CAUSE STACKUNDERFLOW;
```

Für die benutzerdefinierten Ausnahmen ist diese Anweisung erforderlich; für die vordefinierten ist die CAUSE-Anweisung, z. B. für Testzwecke, sehr nützlich.

Wenn an einer Anweisung oder einem Block für eine bestimmte Ausnahme keine Behandlung angefügt ist, dann wird, wie in Ada, die Kette der dynamischen Vorgänger durchsucht, bis entweder eine passende Behandlung gefunden wird oder der äußerste Prozeß (siehe Abschnitt 3i) erreicht ist. Im letzten Falle wird das Programm beendet.

(wird fortgesetzt)

Danksagung

Für wertvolle Hinweise zu früheren Fassungen des Aufsatzes bin ich *W. Hoyer*, *T. Mehner*, *H. Morgenbrod*, *E. Reithmaier* und *H. Sorgenfrei* zu Dank verpflichtet.

Schrifttum

[1] *Martin, T.:* Das Problem, eine Programmiersprache für den Praktiker zu beschreiben. Regelungstechnische Praxis 26 (1984), H. 8, S. 360–362.

[2] CCITT: 4th Plenary Assembly, White Book, Vol. 6, 1969.

[3] CCITT: 5th Plenary Assembly, Green Book, Vol. 6-4, 1973.

[4] CCITT Study Group XI: Blue Document. Proposal for a Recommendation for a CCITT High Level Programming Language (CHILL), 1977.

[5] CCITT: High Level Language (CHILL). Recommendation Z.200, Geneva 1981.

[6] CCITT: CHILL Language Definition, Recommendation Z.200, Geneva 1985.

[7] Reference Manual for the Ada Programming Language. ANSI/MIL-STD 1815A, United States Dept. of Defense, Washington, Jan. 1983.

[8] *Mitchell, J.G., Maybury, W.,* und *Sweet, R.:* Mesa Language Manual. Xerox Palo Alto Research Center, Palo Alto 1979, Version 5.0. CSL-79-3.

[9] *Goldberg, A.,* und *Robson, D.:* Smalltalk-80 – The Language and its Implementation. Addison Wesley Publ. Comp., Reading Mass. 1983.

[10] Siemens AG: EWSD Digital Switching System. telcom report Vol. 4 (1981) Special Issue.

[11] *Winkler, J.F.H.:* Das Prozeßkonzept in Betriebssystemen und Programmiersprachen I, II. Informatik Spektrum 2 (1979), H. 4, S. 219–229, 3 (1980), H. 1, S. 31–40.

[12] *Hoare, C.A.R.:* Monitors: An Operating System Structuring Concept. CACM 17 (1974), H. 10, S. 549–557.

[13] *Brinch Hansen, P.:* Concurrent Pascal Report. Information Science, California Institute of Technology, June 1975.

[14] *Winkler, J.F.H.:* The realisation of data abstractions in CHILL. Third CHILL Conference, Cambridge 1984-09-23 ... 28, S. 175–181.

[15] ISO (International Organization for Standardization): Programming Languages – PASCAL. ISO/DIS 7185, 1982-08-12.

[16] ECMA (European Computer Manufacturers Association): Standard ECMA-50, Programming Language PL/I, December 1976.

[17] *Bjørner, D.,* und *Jones, C.B.:* The Vienna Development Method: The Meta-Language. Springer, Berlin 1978.

[18] CCITT: CHILL Formal Definition. Geneva, December 1981.

[19] *Zigterman, L.:* Specification and Design of an Interlocking System. Third CHILL Conference, Cambridge 1984-09-23 ... 28, S. 161–168.

[20] N.N.: CHILL-Bulletin 4 (1984), H. 2, S. 36–42.

[21] *Rieder, P.,* und *Stadel, M.:* Die Programmiersprache PASCAL. Automatisierungstechnische Praxis 27 (1985), H. 9, S. 435–442.

[22] *Winkler, J.F.H.:* Some Improvements of ISO-PASCAL. SIGPLAN Notices 19 (1984), H. 9, S. 49–62.

[23] *Scheub, V.:* Die Echtzeitprogrammiersprache PEARL. Automatisierungstechnische Praxis 27 (1985), H. 2, S. 79–86.

[24] CCITT: Introduction to CHILL. Geneva 1980.

[25] CCITT: CHILL User Manual 1985.

[26] N.N.: CHILL User Manual. CHILL-Bulletin 4 (1984), H. 1.

[27] *Smedema, C.H., Medema, P.,* und *Boasson, M.:* The Programming Languages Pascal, Modula, CHILL and Ada. Prentice Hall, Englewood Cliffs 1983.

[28] *Martucci, R.:* Introduzione al linguaggio CHILL. Consiglio Nazionale delle Ricerche, Progetto Finalizzato Informatica, Torino 1982.

[29] *Sammer, W.,* und *Schwärtzel, H.:* CHILL – Eine moderne Programmiersprache für die Systemtechnik. Springer, Berlin 1982.

[30] N.N.: CHILL-Bibliographie. CHILL-Bulletin 4 (1984), H. 2, S. 23–35.

[31] *Winkler, J.F.H.:* Schleifen und strukturierte Programmierung. Elektron. Rechenanlagen 18 (1976), H. 4, S. 172–179.

[32] *Wirth, N.:* MODULA-2. ETH Zürich, Inst. für Informatik, Bericht 36, March 1980.

[33] *Mehner, Th.,* und *Winkler, J.F.H.:* An Implementation of the new CHILL-I/O. Third CHILL Conference, Cambridge 1984-09-23 ... 28, S. 195–198.

[34] *Winkler, J.F.H.:* Zum Begriff des Prozesses: am Beispiel von PEARL. Elektron. Rechenanlagen 20 (1978), H. 6, S. 277–282.

Dr. rer. nat. J.F.H. Winkler, Siemens AG, ZTI SOF 213, Otto-Hahn-Ring 6, D-8000 München 83.

Der Weg vom Quellenprogramm (Aufgabenstellung) zum Objektprogramm (Anweisungsliste) muß bei der Ablauftabellentechnik über einen zielsystemspezifischen Compiler führen. Das im Arbeitskreis konzipierte Programmentwicklungssystem soll geeignete Softwareschnittstellen besitzen, an denen diese Compiler angebunden werden können. Die Umwandlung einer Ablauftabelle in den Funktionsplan bei Ablaufsteuerungen und in den Logikplan bei Verknüpfungssteuerungen muß ebenfalls in der Systemsoftware enthalten sein. Wie beim Funktionsplan erfolgt auch bei der Ablauftabelle die Zuordnung von Prozeßvariablen zu diskreten Ein/Ausgängen des Zielsystems in der auch später erstellbaren E/A-Liste, ebenso wie die Zuordnung von Zählereingängen zu Zählerbausteinen.

(wird fortgesetzt)

Schrifttum

[1] *Bessai, B.:* Strichsymbolik in Datenfluß- und Programmablaufplänen. Online-adl-Nachrichten 6 (1977), S. 479–483.

[2] *Franke, B.:* Strichpläne zur Kommunikation in der Softwaretechnologie. ÖVD/Online 3 (1982), S. 70–73.

[3] *Hengstenberg, J., Sturm, B.,* und *Winkler, O.:* Messen, Steuern und Regeln in der Chemischen Technik. Band III, 3. Auflage, Springer Verlag 1981.

[4] *Coblenz, R.:* Entwurf und Realisierung von Steuerungen mit Hilfe von Ablauftabellen. Bayer-interner Bericht.

Arbeitsbericht (NAMUR-Report) des NAMUR-Arbeitskreises „Bildschirmgestützte Programmentwicklung und Dokumentation von Verknüpfungs- und Ablaufsteuerungen" im NAMUR-Ausschuß „Prozeßleitsysteme".

Dipl.-Ing. S. Weidlich, Hoechst AG, EMR-Technik/Betrieb, D 710, Postfach 800320, D-6230 Frankfurt am Main 80.

Die Programmiersprache CHILL

J. F. H. Winkler, München

(Teil 2, Fortsetzung von Heft 5/1986)

Die Programmiersprache CHILL wurde in der Zeit von 1975 bis 1983 vom CCITT zum Zwecke der Programmierung von Vermittlungsrechnern entwickelt. Vermittlungsrechner sind Prozeßrechner, und daher enthält CHILL auch Sprachelemente, die für die Programmierung von Prozeßrechnern wichtig sind. Beispiele hierfür sind nebenläufige Prozesse und Darstellungsspezifikationen für Daten. Die Programme von Vermittlungssystemen sind außerdem sehr umfangreich und komplex. Daher enthält CHILL auch Sprachelemente für das Programmieren im Großen, und zwar ein sehr flexibles Modulkonzept.

In der vorliegenden Arbeit wird CHILL in Anlehnung an den Vorschlag [1] dargestellt.

f) Datenstrukturen

Wie alle höheren Programmiersprachen enthält CHILL skalare und strukturierte Datenarten. Wie in Algol 68 werden Datentypen in CHILL als „modes" bezeichnet.

Die **skalaren Datenarten** von CHILL sind:

- INT: ein Teilbereich der ganzen Zahlen. Eine Implementierung darf weitere Ganzzahltypen, wie z. B. LONGINT und SHORTINT, definieren. Ganzzahlkonstanten können dezimal, binär, oktal oder hexadezimal notiert werden.
- BOOL: der Typ der logischen Wahrheitswerte mit den Werten FALSE und TRUE.
- CHAR: die Zeichen des CCITT Alphabetes Nr. 5.
- *Aufzählungstypen.* Der Programmierer kann Aufzählungstypen mit eigenen Werten definieren. Die Typbeschreibung ist eine geordnete Menge von Werten:

```
SYNMODE Gerätezustand =
  SET (Frei, Belegt, Defekt);
```

Durch diese Definition wird ein Typ „Gerätezustand" vereinbart, dessen Wertevorrat aus den drei Werten „Frei", „Belegt", und „Defekt" besteht. Pascal-Kenner müssen sich davor hüten, den Gebrauch des Schlüsselwortes SET in CHILL mit dem in Pascal zu verwechseln. (SET in Pascal entspricht POWERSET in CHILL). Wie in anderen Programmiersprachen auch, sind die Werte eines Aufzählungstyps im Sinne der Aufschreibung geordnet.

Für Automatisierungsanwendungen kann es wichtig sein, die interne Codierung der Werte eines Aufzähltyps im Programm zu spezifizieren. Dies ist z. B. dann wichtig, wenn diese Werte durch die Apparatur (Hardware) definiert sind, wie z. B. die Inhalte von Geräteregistern. Dazu gibt es in CHILL eine zweite Form der Definition eines Aufzählungstyps:

```
SYNMODE Gerätezustand =
    SET ( Frei    = 2,
          Belegt  = 5,
          Defekt  = 6 );
```

- *Bereichstypen.* Von all den bisher genannten Typen können Teilintervalle als neue Typen definiert werden:

```
SYNMODE Positiv = INT (0 : MaxInt);
```

- *Mengentypen.* Wie in Pascal können auch in CHILL Mengentypen über diskreten Basistypen definiert werden:

```
SYNMODE ZeichenMengen =
POWERSET CHAR;
```

Der Wertevorrat von „ZeichenMengen" enthält alle Teilmengen von CHAR. An eine Variable von diesem Typ können dann entsprechende Mengen zugewiesen werden:

```
DCL GrossBuchstZeichenMengen := ['A' : 'Z'];
```

Für die Mengentypen gibt es folgende Operationen: Vereinigung, Durchschnitt, Teilmengenrelation, Elementrelation, Differenz, symmetrische Differenz und Komplementbildung.

- *Verweistypen.* In CHILL gibt es sowohl die üblichen typgebundenen Verweise als auch typungebundene Verweise.
- *Unterprogrammtypen.* Wie in Algol 68 gibt es auch in CHILL Unterprogrammtypen. Diese stellen eine Fortentwicklung der formalen Unterprogramme in Algol 60 und Pascal dar. Ein Beispiel für die Definition eines Unterprogrammtyps ist:

```
SYNMODE TreiberTyp =
    PROC ( Daten REF Datenbeschreibung IN,
           Richtung Richtungsangabe IN )
        EXCEPTIONS ( GeraetBelegt,
                     GeraetDefekt );
```

Die Unterprogrammtypen können wie andere Typen verwendet werden, z. B. in der Definition von Variablen, Parametern und neuen Typen. Die Werte von Unterprogrammtypen sind die im Programm definierten Unterprogramme. Der oben definierte Unterprogrammtyp kann z. B. folgendermaßen verwendet werden:

```
SYNMODE GeraeteBeschreibung =
  STRUCT (Art       GeraeteArt,
          Zustand   GeraeteZustand,
          Treiber   TreiberTyp,
          CASE Art OF
              (Platte): AnzahlZylinder...
              (Drucker): ....
          ESAC);
```

Die **zusammengesetzten Datenarten** von CHILL sind die folgenden:

- *Zeichenkettentypen.* Es gibt zwei Arten von Zeichenkettentypen:

CHAR(Länge) und BIT(Länge),

wobei „Länge" ein Ausdruck mit positiv ganzzahligem Wert ist. Die erste Form ist für Zeichenketten, deren Komponenten Zeichen des Typs CHAR sind, und die zweite Form für Zeichenketten, deren Komponenten die Zahlen 0 oder 1 sind. Diese zweite Form wird im Folgenden als Bitkette bezeichnet.

Zeichenketten können in Klartextform oder in Hexadezimalform formuliert werden:

'CHILL' und C'43_48_49_4C_4C'.

Bei der zweiten Schreibweise darf der Unterstreichungsstrich als Gliederungshilfsmittel verwendet werden.

Bitketten können binär, oktal oder hexadezimal formuliert werden:

B'1011_0010' = H'B2'.

Werte eines Zeichenkettentyps können entsprechend der lexikografischen Ordnung miteinander verglichen werden. Zeichenketten einer Art können mit dem Verkettungsoperator verkettet werden:

'CH'//'C'49'//'LL'<'DABEI'.

- *Reihungstypen.* Es gibt Reihungstypen mit beliebig vielen Dimensionen und beliebigem Komponententyp. Wie in Pascal ist

```
ARRAY (1 : 20, 1 : 10) INT
```

äquivalent zu

```
ARRAY (1 : 20) ARRAY (1 : 10) INT.
```

- *Verbundtypen.* Die Verbundtypen sind denen in Modula-2 [32] sehr ähnlich. Von denen anderer Sprachen unterscheiden sie sich dadurch, daß jede Komponente eine Komponente mit Varianten sein kann. In den anderen höheren Programmiersprachen kann in der Regel nur die letzte Komponente eine mit Varianten sein. Ein Beispiel für einen Verbundtyp in CHILL ist:

```
STRUCT ( Form FigurTyp,
         Position PosTyp,
         CASE Form OF
           (Quadrat)   : Seite RANGE (0:100),
           (Kreis)     : Radius RANGE (0:50),
           (Rechteck)  : Hoehe,
                         Breite RANGE (0:100)
         ESAC,
         Fuellung SET (Schraffur, Farbe),
         CASE Fuellung OF
           (Schraffur) : SchraffArt SchraffTyp,
           (Farbe)     : FarbArt FarbTyp
         ESAC )
```

Analog zur CASE-Anweisung kann die Variantenauswahl auch über mehrere Dimensionen erfolgen:

```
STRUCT (tag1 Color,
        tag2 BOOL,
        CASE tag1,     tag2 OF
             (red),    (TRUE):   k1 T1,
             (green),  (TRUE):   k2 T2,
             (blue),   (TRUE):   k3 T3,
             (red),    (FALSE):  k4 T4,
             (green),  (FALSE):  k5 T5,
             (blue),   (FALSE):  k6 T6
        ESAC);
```

Für Reihungstypen und Verbundtypen können Angaben über die Abbildung auf den Speicher gemacht werden. Damit läßt sich die unterliegende Apparatur mit den Mitteln der Programmiersprache beschreiben.

g) Ein- und Ausgabeverkehr ohne Echtzeitbezug

CHILL bietet sequentiellen und wahlfreien Zugriff auf Dateien. Die Zuordnung einer externen Datei zu einem internen Platzhalter kann ebenfalls im Programm ausgedrückt werden und dynamisch zur Ablaufzeit durchgeführt werden. Damit ist eine beliebige Zuordnung von Dateien zur Laufzeit möglich [33]. Alle Datentypen können als Typ der Dateikomponenten auftreten.

h) Echtzeiteigenschaften

Zur Formulierung von Nebenläufigkeiten stehen in CHILL Prozesse und verschiedene Synchronisations- und Kommunikationsmittel zur Verfügung. In seinem Aufbau ähnelt ein Prozeß einem Unterprogramm:

```
Proz: PROCESS ();
    wait : PROC (x INT);
          /* eine Warteaktion */
    END wait;
    DO FOR EVER;
          wait (10 /* Sekunden */);
    OD;
END Proz;
```

Ein Prozeß wird gestartet durch einen START-Ausdruck:

```
START Proz ().
```

Hat der Prozeß Parameter, dann werden im START-Ausdruck entsprechende aktuelle Parameter angegeben.

Die Auswertung eines START-Ausdruckes erzeugt eine neue Inkarnation [34] der im START-Ausdruck genannten Prozeßdefinition und startet diese neue Inkarnation. Sind Parameter vorhanden, dann werden diese berechnet und an die neugeschaffene Prozeßinkarnation übergeben. Die neugeschaffene Prozeßinkarnation läuft anschließend parallel zu den bereits vorhandenen Prozeßinkarnationen ab, insbesondere auch zu der, von welcher sie gestartet wurde. Das Ergebnis eines START-Ausdruckes ist ein eindeutiger Wert vom Typ INSTANCE. Dieser Wert identifiziert die neugeschaffene Inkarnation eindeutig und kann anschließend verwendet werden, um diese neue Inkarnation anzusprechen.

Zu diesem Zweck kann man den Wert des START-Ausdruckes an eine Variable vom Typ INSTANCE zuweisen:

```
WarteProz := START Proz ();
```

Eine Prozeßdefinition in CHILL entspricht der Definition eines Tasktyps in Ada [7]. Im Unterschied zu PEARL können in CHILL mehrere Inkarnationen eines Prozesses gleichzeitig aktiv sein.

Ein Prozeß wird beendet, wenn er das Ende seines Blockes erreicht, oder wenn er eine STOP-Anweisung ausführt. Das Programm wird frühestens dann beendet, wenn alle darin gestarteten Prozeßinkarnationen beendet sind.

Der Zugriff mehrerer Prozesse zu gemeinsamen Daten kann über Monitore koordiniert werden. Das folgende Beispiel zeigt eine einfache Pufferverwaltung.

```
PufferVerwalter:
   REGION
      GRANT Lesen, Schreiben, Txy;
      NEWMODE Txy = ...;
      SYN PufferGroesse = 100;
      NEWMODE PIndex = INT (1: PufferGroesse);
      DCL Puffer ARRAY ( PIndex ) : = 0;
      DCL AnzahlVolle INT(0:PufferGroesse) : = 0;
      DCL EinInd, AusInd PIndex : = 1;
      DCL Voll, Leer EVENT;
      Lesen: PROCEDURE (EL Txy OUT);
         IF AnzahlVolle = 0
         THEN DELAY Leer; FI;
         EL: = Puffer (AusInd);
         AusInd: = (AusInd MOD PufferGroesse) + 1;
         AnzahlVolle -: = 1;
         CONTINUE Voll;
      END Lesen;
      Schreiben: PROCEDURE (EL Txy IN);
         IF AnzahlVolle = PufferGroesse
         THEN DELAY Voll; FI;
         Puffer (EinInd): = EL;
         EinInd: = (EinInd MOD PufferGroesse) + 1;
         AnzahlVolle + : = 1;
         CONTINUE Leer;
      END Schreiben;
END PufferVerwalter;
```

Wie bei einem Monitor kann zu einem Zeitpunkt höchstens eines der beiden Unterprogramme „Lesen" oder „Schreiben" aufgerufen sein. DELAY bewirkt ein Einreihen in die Warteschlange des genannten EVENT.

CONTINUE entnimmt den ersten Prozeß aus der Warteschlange des genannten EVENT und setzt ihn fort. Wenn die Warteschlange leer ist, dann hat die CONTINUE-Anweisung keine Wirkung.

Zur Kommunikation zwischen Prozessen dienen BUFFER und SIGNAL. Mittels BUFFER können Puffer mit einer vorgebbaren Länge und beliebigem Komponententyp definiert werden. Mit der Anweisung

```
SEND Puffer (Ausdruck);
```

kann der Wert von „Ausdruck" an „Puffer" geschickt werden. Die mittels BUFFER definierten Puffer arbeiten im Prinzip nach dem Protokoll, wie es oben im Monitor „Puffer-Verwalter" dargestellt wurde (FIFO). Außerdem kann in der Sendeanweisung noch eine Prioritätsangabe gemacht werden. Dadurch ergibt sich dann ein von FIFO abweichendes Protokoll.

Zum Empfangen von Daten aus einem BUFFER führt ein Prozeß eine RECEIVE-Anweisung aus. Dabei kann auch alternativ aus einem von mehreren Puffern empfangen werden. Diese Kontruktion ist ähnlich zur SELECT-Anweisung in Ada.

Ein zweites Hilfsmittel für die Prozeßkommunikation ist das SIGNAL. Dabei handelt es sich um einen Kommunikationskanal, der *einen* Verbund (Record) beliebigen Typs aufnehmen kann. Die SEND-Anweisung schreibt einen Wert in das SIGNAL. Die RECEIVE-Anweisung liefert den seit der vorhergehenden RECEIVE-Anweisung zuletzt durch eine SEND-Anweisung in das SIGNAL geschriebenen Wert. Wurde seit dem letzten RECEIVE kein entsprechendes SEND ausgeführt, dann wartet der das RECEIVE ausführende Prozeß auf ein solches SEND.

Beispiel:

```
SIGNAL S1 = (INT, BOOL) TO Proz;
SEND S1 (15, TRUE) TO InkarnationsId;

RECEIVE CASE SET InkarnationsVar;
   (S1 IN VI, VB):   Aktion1;
   (S2...):          Aktion2;
ESAC;
```

Das Signal S1 besteht aus zwei Komponenten mit den Typen INT und BOOL. Nur Prozesse des Typs „Proz" können Daten von diesem Signal empfangen.

Die SEND-Anweisung sendet die Werte „15" und „TRUE" an das Signal „S1". Außerdem können diese Daten nur von dem Prozeß empfangen werden, dessen Prozeßidentifikation im „InkarnationsId" enthalten ist. Dieser Prozeß muß außerdem vom Typ „Proz" sein.

In der RECEIVE-Anweisung wird alternativ auf die Signale „S1" und „S2" gewartet. „S2" ist ein reines Synchronisationssignal, da es keine Datenkomponenten enthält. Die Prozeßidentifikation des Prozesses, von dem empfangen wurde, wird an „InkarnationsVar" zugewiesen.

Wird „S1" empfangen, dann wird „Aktion1" ausgeführt und entsprechend bei „S2" die „Aktion2".

Die RECEIVE-Anweisungen können auch einen ELSE-Zweig enthalten, der dann ausgeführt wird, wenn unmittelbar von keinem der genannten Signale ein Empfang möglich ist.

i) Programmstruktur

Ein CHILL-Programm ist eine Folge von Modulen und Monitoren:

{Modul|Monitor}+ .

Diese Folge hat man sich als Rumpf eines äußersten, imaginären Prozesses vorzustellen, der durch einen Mechanismus außerhalb des Programmes gestartet wird. Blöcke dürfen weitgehend freizügig geschachtelt werden, wie man das von den Sprachen mit Blockstruktur her gewohnt ist. Es bestehen jedoch einige Ausnahmen:

- Prozesse können nur in Modulen vereinbart werden; sie können nicht geschachtelt werden und nicht in Unterprogrammen enthalten sein.
- Monitore können nur auf oberster Programmebene definiert werden; sie können nicht geschachtelt werden und nicht in anderen Blöcken enthalten sein.

Bei der Schachtelung von Blöcken gilt die seit Algol 60 wohlbekannte globale Sicht, das heißt, in eingeschachtelten Blöcken, die keine Module sind, sind die in umgebenden Blöcken definierten Größen im allgemeinen automatisch sichtbar. Bei Modulen und Monitoren ist es darüber hinaus möglich, auf darin definierte Größen von außen zuzugreifen. Zu diesem Zweck müssen solche Größen mit einer GRANT-Anweisung exportiert werden:

```
GRANT Lesen, Schreiben, Txy;
```

Der Export von Größen erfolgt im Prinzip wie bei Modula-2, das heißt, es werden einzelne Größen exportiert. Die Exportanweisung in CHILL ist allerdings wesentlich vielseitiger: es können auch alle Größen en bloc exportiert werden:

```
GRANT ALL;
```

und es können beim Exportieren frei gewählte Präfixe vor die Namen der exportierten Größen gesetzt werden:

```
GRANT Lesen, Schreiben PREFIXED Puffer;
```

In der Umgebung des betreffenden Moduls können die Größen „Lesen" und „Schreiben" dann durch „Puffer!Lesen" bzw. „Puffer!Schreiben" angesprochen werden.

Andere Module und Monitore müssen Größen, die durch Export in ihrer Umgebung sichtbar gemacht werden, im allgemeinen explizit durch Importanweisungen importieren. Davon sind solche Größen ausgenommen, die mit dem Attribut PERVASIVE exportiert werden.

Eine einfache Importanweisung hat die Form

```
SEIZE Lesen, Schreiben;
```

Werden Größen importiert, die mit einem Präfix exportiert wurden, dann kann dieses Präfix beim Import umbenannt werden:

```
SEIZE (Puffer ⇒ PV) ! Lesen;
```

Anschließend muß die importierte Größe mit „PV!Lesen" angesprochen werden.

Das Präfix kann auch zum Verschwinden gebracht werden, z. B. durch

```
SEIZE (Puffer ⇒ ) ! Lesen;
```

Anschließend kann „Lesen" direkt verwendet werden, um die externe Größe „Puffer!Lesen" anzusprechen.

Es besteht außerdem die Möglichkeit, den Modul- oder Monitornamen automatisch als Präfix beim Export zu verwenden:

```
GRANT Lesen, Schreiben PREFIXED;
```

Außerhalb des Moduls „PufferVerwalter" müssen die exportierten Größen dann mit „PufferVerwalter!Lesen" bzw. „PufferVerwalter!Schreiben" angesprochen werden.

Diese Möglichkeiten ergeben insgesamt ein flexibles Konzept für den Ex- und Import von Namen zwischen Modulen und Monitoren.

j) Effizienz

Die Effizienz der Objektprogramme wird im wesentlichen durch den Übersetzer und das Laufzeitsystem bestimmt. An Vermittlungsrechner werden bezüglich der Reaktionsgeschwindigkeit und der zu bewältigenden Verkehrslast hohe Anforderungen gestellt. Diese konnten mit CHILL erfüllt werden.

Subsets von CHILL wurden auch auf verschiedenen Mikroprozessoren implementiert. Auch die volle Sprache läßt sich, wie z. B. auch Ada, auf modernen Mikroprozessoren implementieren.

k) Sprachumfang

Wie bereits in [1] erwähnt, kommt es besonders bei umfangreichen Sprachen – zu welchen CHILL sicherlich gehört – vor, daß nur Teilmengen der genannten Sprache implementiert werden. Dieser Fall ist auch bei CHILL eingetreten, da einige Implementierungen bereits in der letzten Phase der Sprachentwicklung vorgenommen wurden, und anschließend relativ rasch umfangreiche Produkte mittels CHILL realisiert wurden (siehe Einleitung). Diese Situation ist vom Standpunkt der reinen Sprachlehre aus gesehen nicht ideal, wirkt sich bei den bisherigen Anwendungen aber noch nicht so gravierend aus, da dabei Portabilität zwischen beliebigen Rechnern nicht erforderlich ist. Da sich mittlerweile die Sprache weitgehend stabilisiert hat, dürfte sich die Portabilität in Zukunft nur verbessern.

Dr. rer. nat. J.F.H. Winkler, Siemens AG, ZTI SOF 213, Otto-Hahn-Ring 6, D-8000 München 83.

Ein PEARL-PC mit Z80-Mikroprozessor

„Vergleichbare Computer-Leistung kostet nach zehn Jahren ein Zehntel", sagt eine Faustregel. Verglichen mit durchschnittlichen Prozeßrechnern aus der Mitte der sechziger Jahre, haben Personal Computer mit Z80-Mikroprozessor heute etwa gleiche Rechenleistung und größere Haupt- und Massenspeicher, kosten aber weniger als ein Hundertstel. Wer diesen Preisvorteil nutzen will, sollte sich jedoch auch der Fortschritte in der Softwaretechnik bedienen und nicht mehr in Assembler programmieren.

Am Universitäts-Rechenzentrum in Erlangen ist deshalb schon vor einigen Jahren ein PEARL-System für den Z80 entwickelt worden. Es enthält ein sprachbezogenes Testsystem für den Laufzeittest und kann auch für die Programmierung von verteilten Systemen verwendet werden. Neben dem Einsatz in der Lehre ist es in mehreren Automatisierungs-Vorhaben innerhalb der Universität und in Forschungszentren erfolgreich erprobt worden.

Das System ist in der PEARL-Spalte von rtp-Heft 9/1983 beschrieben worden, um einen Technologie-Transfer anzuregen. Der ist inzwischen erfolgt: seit einigen Monaten wird es von der Heusch-Bösefeld GmbH vertrieben und gepflegt und in eigenen und Kundenprojekten eingesetzt. Die Firma bietet die Entwicklung kompletter Anwendersysteme an und liefert für den preiswerten PEARL-Einsatz in der Labor- und Prozeßautomatisierung den Rechner HBS80 (Z80-Zentraleinheit, Arithmetikprozessor AMD 9511, ECB-Bus). Die Hardware ist modular aufgebaut; es gibt Karten für interruptgesteuerte Standardperipherie, Prozeß-Ein/Ausgabe und eine IEC-Bus-Kopplung, die vom PEARL-System voll unterstützt werden. Verfügbar ist außerdem ein in PEARL geschriebenes, GKS-kompatibles Graphikpaket.

Inzwischen brauchen auch keine Hostrechner (CYBER, VAX) mehr verwendet zu werden, sondern die PEARL-Programme können auf dem PC selbst unter dem Betriebssystem CP/M kompiliert werden. Beim Start eines PEARL-Programmes wird das CP/M-Betriebssystem durch das PEARL-Echtzeit-Betriebssystem überladen, auf dem auch das sprachbezogene Testsystem aufsetzt. Unter PEARL erstellte Dateien sind CP/M-kompatibel.

Der Programmierkomfort von PEARL verfügt erfahrungsgemäß dazu, die Software von vornherein so breit anzulegen, daß der zur Verfügung stehende Speicher schnell ausgeschöpft ist. Das PEARL-Betriebssystem für den Z80 ist deshalb dergestalt modular aufgebaut, daß nicht benötigte Funktionen problemlos gestrichen werden können. Sein Kern belegt nur etwa 4 kbyte; pro Gerätetreiber werden 0,5 bis 2 kbyte benötigt, für die PEARL-Laufzeitroutinen 2 bis 20 kbyte.

Auskünfte erteilt: Heusch-Bösefeld GmbH, Liebigstraße 20, D-5100 Aachen, Telefon (0241) 165001 (Herr Dipl.-Ing. *Dowideit* oder Herr Dipl.-Ing. *Schirk*).

L. Frevert

Prof. Dr. L. Frevert, Ostersiek 29 D-4902 Bad Salzuflen 1.