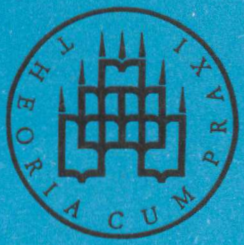
Rostocker Mathematisches Kolloquium

Heft 14



WILHELM-PIECK-UNIVERSITÄT
ROSTOCK

ROSTOCKER MATHEMATISCHES KOLLOQUIUM

Heft 14

Arbeitsseminar über theoretische Grundlagen von Dialogsystemen

Rostock, 27. - 28. 09. 1979

1980

Wilhelm-Pieck-Universität Rostock

Sektion Mathematik

Herausgeber: Der Rektor der Wilhelm-Pieck-Universität Rostock

Schriftleitung: Prof. Dr. Wolfgang Engel, Direktor der Sektion Mathematik
Doz. Dr. Gerhard Maeß, Schriftleiter
Dr. Werner Plischke, Lektor
Dorothea Meyer, Herstellung der Druckvorlage

Sektion Mathematik der Wilhelm-Pieck-Universität Rostock,
DDR - 2500 Rostock, Universitätsplatz 1

Das Rostocker Mathematische Kolloquium erscheint in der Regel dreimal im Jahr und ist im Rahmen des Schriftentausches über die Universitätsbibliothek, Tauschstelle, DDR - 2500 Rostock, Universitätsplatz 5, zu beziehen.

Veröffentlicht durch die Abt. Wissenschaftspublizistik der Wilhelm-Pieck-Universität Rostock, DDR - 2500 Rostock,
Vogelsang 13/14 Fernruf: 369 577
Leiter: Dipl.-Ges.-Wiss. Bruno Schrage
Genehmigungs-Nr. C 79/80
Druck: Ostsee-Druck Rostock, Werk II

Inhalt

Vorwort

Am 27. und 28. September 1979 fand am Rechenzentrum der Wilhelm-Pieck-Universität Rostock ein Arbeitsseminar über theoretische Grundlagen von Dialogsystemen statt. Teilnehmer waren wissenschaftliche Mitarbeiter von Universitäten, Hochschulen, der Akademie und Betrieben der DDR. Neben Vorträgen, vorbereiteten Diskussionbeiträgen, zu denen der Veranstalter gezielt aufgefordert hatte, spielte die Diskussion zur Herausarbeitung von Standpunkten und praktischen Anwendungen zur genannten Problematik eine wesentliche Rolle.

Zentrales Anliegen des Arbeitsseminars war die Vorstellung und Diskussion von Modellen, insbesondere von mathematischen Modellen für Dialogsysteme oder für Teile von Dialogsystemen. Über die Notwendigkeit mathematischer Modelle als Grundlage der Entwicklung, Projektierung, Implementierung und Bewertung von Dialogsystemen waren sich alle Teilnehmer einig. Die praktische Verwendbarkeit einzelner Ansätze ließ sich hingegen weniger klar nach einheitlichen Gesichtspunkten einschätzen. Die Standpunkte, die zur Entwicklung von Modellen führen und unterschiedliche Ziele verfolgten, waren naturgemäß sehr verschieden. Die Vielfalt von Ansätzen und die nicht immer einheitlich gebrauchten Begriffe zeigten jedoch sehr deutlich die Zweckmäßigkeit derartiger Arbeitsseminare. Der Wunsch nach Veröffentlichung der Vorträge war groß, so daß wir gern von der Bereitschaft zum Druck einiger Vorträge in der "ROMAKO-Reihe" Gebrauch machen.

Auf die Darlegung grundsätzlicher Positionen zur Problematik verzichten wir hier in den Vorbemerkungen und verweisen auf den ersten Teil des Beitrages von K.-H. Kutschke.

Wenn auch die Wiedergabe der Gesamtdiskussion nicht möglich ist, so hoffen wir doch, daß die vorliegenden Beiträge einen Einblick in bestimmte Modelle gewähren und der breiteren Diskussion als Grundlage dienen können.

Für die Möglichkeit der Veröffentlichung von Beiträgen des Arbeitsseminars möchten wir insbesondere Herrn Prof. Dr. sc. nat. W. Engel, Direktor der Sektion Mathematik, und Doz. Dr. sc. nat. G. Maeß, Fachredakteur des Rostocker Mathematischen Kolloquiums, herzlich danken. Unser Dank gilt ebenfalls Herrn Dr. rer. nat. W. Plischke für seine umsichtige Arbeit als Lektor und Frau D. Meyer für das sorgfältige Anfertigen der Druckvorlagen.

Prof. Dr. sc. nat. K.-H. Kutschke
Seminarleiter

Rostock. Math. Kolloq. 14, 7 - 15 (1980)

Bodo Hohberg

Abstrakte Dialogautomaten als Hilfsmittel für den Entwurf von Dialogcompilern

Mit den ständig wachsenden technischen Voraussetzungen für den Dialog zwischen Rechner und Benutzer wachsen auch die Programmieraufgaben, die diesen Dialog ermöglichen sollen. Programmierungstechnische Hilfsmittel für den Entwurf von dialogfähigen Programmen gewinnen daher an Bedeutung.
Diese Arbeit geht davon aus, daß die Konzeption des Dialogs in die Konzeption der übrigen Programmteile zu integrieren ist. Zunächst wird daher ein Automatenmodell definiert, das die zu entwerfende Programmstruktur und den Informationsfluß gleichzeitig ausweist. Die zusätzliche Spezifikation von Dialogaufgaben führt zu einem Dialogautomaten. Der weitere Programmentwurf ergibt aber wieder ein Modell der Ausgangsform, wodurch eine schrittweise Einbeziehung von Dialogaufgaben in den Entwurf möglich wird.

1. Praktischer Hintergrund

Unter einem Dialogcompiler soll, wie weithin üblich, ein Dialogprogrammiersystem verstanden werden, das im Dialog mit einem Benutzer die Programmeingabe, die Compilation, Korrekturen und den Programmtest realisiert. Bei den Untersuchungen in /1/ und der praktischen Realisierung des MS-Algol-Dialogcompilers in /2/ haben sich vor allem zwei Sachverhalte deutlich gezeigt, die für eine auf den Entwurfsprozeß abgestimmte Definition von Dialogautomaten wichtig sind. Als Thesen formuliert sind dies:

1. Die Dialogfunktionen eines Dialogcompilers lassen sich in natürlicher Weise von den übrigen Funktionen trennen. Der Entwurf eines Dialogcompilers vereinfacht sich, wenn Dialogfunktionen und übrige Funktionen voneinander abgehoben werden (Methode des schrittweisen Ausbaus von Programmen).

2. In einem Dialogcompiler sind folgende Dialogfunktionen vorzusehen:

   a) Ausgaben von Informationen an den Benutzer;
   b) Änderung von Daten durch den Benutzer;
   c) Einfluß auf die Fortsetzung der Arbeit des Dialogcompilers.

Die beiden Thesen besagen, daß für den Entwurf von Dialogcompilern der nicht Dialog führende Teil die primäre Rolle spielt, während die Dialogkomponenten eine sekundäre Bedeutung besitzen. Diese Aussage dürfte weithin für Dialogsysteme gelten. Anders ist der Ausgangpunkt jedoch bei Systemen, bei deren Spezifikation die Dialoganforderungen dominieren. Die entworfenen Modelle scheinen aber auch für derartige Aufgaben anwendbar zu sein, da, wie Abschnitt 4 zeigt, nach der Konzeption der spezifizierten Dialogfunktionen ein Modell entsteht, das den Modellen für die nicht Dialog führenden Programmteile entspricht.

2. Informationsfluß spezifizierende Automaten

Zur Beschreibung der Funktion eines Programms eignet sich ein MEALY-Automat /3/

$$A = (X, Y, Z, f, g)$$

mit X Menge der Eingabezeichen, Y Menge der Ausgabezeichen, Z Zustandsmenge, f Abbildung aus $Z \times X$ in Z (Überführungsfunktion) und g Abbildung aus $Z \times X$ in Y (Ergebnisfunktion). Dieses Modell eines Programms ist aber nicht in der Lage, die Programm- und Datenstruktur widerzuspiegeln. Soll ein Automatenmodell in einem Entwurfsprozeß Verwendung finden, so ist die Zustandsmenge Z und die Überführungsfunktion f in einer dem Entwurfsprozeß angepaßten Weise zu strukturieren. Im besonderen sind folgende Prinzipien zu berücksichtigen:

1. Die Implementation der im Modell ausgewiesenen Teilalgorithmen ist ohne Kenntnis der Struktur der übrigen, als Einheit benutzten Teilalgorithmen möglich (Top-down Entwurf).

2. Informationsfluß im Programm und Verarbeitungsaufgaben sind zwei gleichberechtigte Angelpunkte des Entwurfs.

3. Das Automatenmodell muß eine anschauliche Repräsentation besitzen, mit der beim Entwurf hantiert werden kann.

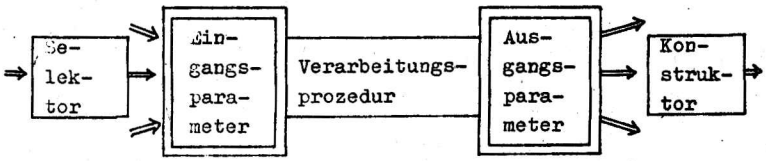
Für die Konzeption von Dialogfunktionen kommen folgende Gesichtspunkte hinzu:

4. Die gesamte, vom Programm verwaltete Information muß für die Spezifikation des Informationsaustauschs beim Dialog bekannt sein.

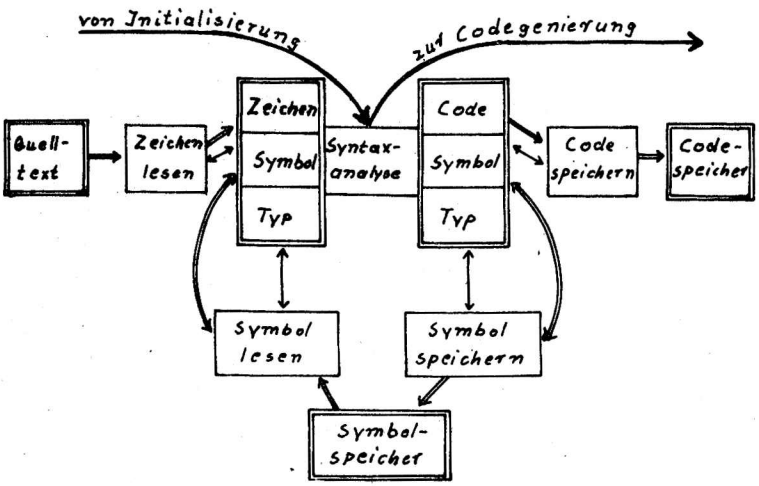
5. Die Fortsetzungsmöglichkeiten nach der Unterbrechung des Programms sind auszuweisen.

Zusammenfassend geht es darum, einen den Informationsfluß spezifizierenden Automaten zu definieren. Die im folgenden vorgenommene Definition basiert auf einem Modell, das Informationsspeicher und Verarbeitungsroutinen unterscheidet. Der Informationsfluß wird durch Zugriffsrechte zu den Informationsspeichern /4/ beschrieben. Die Zugriffsfunktionen werden nach /5/ in Selektoren (Bereitstellen von Informationen zur weiteren Verarbeitung) und Konstruktoren (Verändern von für wiederholten Zugriff gespeicherter Information) eingeteilt. Eigene Untersuchungen regten an, Verarbeitungsprozeduren streng von den Selektoren und Konstruktoren zu trennen. (Bei der praktischen Programmierung erwies es sich sogar als günstig, Selektoren und Konstruktoren auf die Auswahl von Daten, Datentransporte und das Löschen von Daten zu beschränken.)

Die prozeduralen Bausteine des Modells haben folgende Form:



Selektoren und Konstruktoren definieren die Schnittstelle zwischen globalem Datenspeicher und den Verarbeitungsprozeduren. Die interne Struktur aller Prozeduren ist für das Modell unwesentlich. Eine andere Sache ist, daß der Informationsbedarf einer Verarbeitungsprozedur oft erst bei einem folgenden Top-down-Entwurfsschritt offensichtlich wird und das Modell im Laufe des Entwurfs eine Verfeinerung erfahren kann. Als Beispiel diene das Modell des Syntaxanalyseteils eines Compilers, in dem die Verbindungen zum übrigen Teil des Compilers offen gelassen sind.



In diesem Bild des Modells weisen die einfachen Linien den Programmablauf aus. Doppellinien zeigen den Informationsfluß. Datenspeicher besitzen einen doppelten Rand. Die Eingabe in das Programm erscheint wie die Aktivierung einer Selektorfunktion. Die Definition eines den Informationsfluß spezifizierenden Automaten hat die folgende Form.

Definition 1: Ein den Informationsfluß spezifizierender Automat A_I ist ein Tripel

$A_I = (A, S, F)$ mit:

$A = (X,Y,Z,f,g)$ ist ein MEALY-Automat;

$S = (L,G)$ ist die Speicherstruktur von A mit

$L = L_1 x \ldots x L_m$; $L_1,\ldots,L_m$ Wertemengen (lokaler Speicher),

$G = G_1 x \ldots x G_n$; $G_1,\ldots,G_n$ Wertemengen (globaler Speicher),

$Z = L x G$;

$F = (V,S,K,p)$ ist die Funktionsstruktur mit:

$V = \{v_1,\ldots,v_x\}$; $v_1,\ldots,v_x$ sind Abbildungen aus $Z \times X$ in Z, die sich auf isomorphe Abbildungen aus L in L einschränken lassen (Verarbeitungsprozeduren);

$S = \{s_1,\ldots,s_y\}$; $s_1,\ldots,s_y$ sind Abbildungen aus $Z \times X$ in Z, die sich auf isomorphe Abbildungen aus $Z \times X$ in L einschränken lassen (Selektoren);

$K = \{k_1,\ldots,k_z\}$; $k_1,\ldots,k_z$ sind Abbildungen aus $Z \times X$ in Z, die sich auf isomorphe Abbildungen aus Z in G einschränken lassen (Konstruktoren);

p ist eine Abbildung aus Z in $V \cup S \cup K$, so daß für $z \in Z$ und $x \in X$ $f(z,x) = p(z)(z,x)$ (Positionsfunktion) gilt.

Ein den Informationsfluß spezifizierender Automat wird im folgenden kurz I-Automat genannt.
Im Beispiel sind die angegebenen Teile des Compilers wie folgt den Elementen eines I-Automaten zuzuordnen:

Speicherstruktur

$S = (\{$Zeichen, Symbol, Typ, Code,...$\}$,
$\{$Symbolspeicher, Codespeicher,...$\})$

Funktionsstruktur

$F = (\{$Syntaxanalyse,...$\}$,
$\{$Zeichen lesen, Symbol lesen,...$\}$,
$\{$Symbol speichern, Code speichern,...$\}$,
p (ist im Bild nicht vollständig definiert)).

Das Bild des I-Automaten wird im besonderen dadurch übersichtlich, daß den Prozeduren nur die jeweils benötigten Speicherkomponenten zugeordnet wurden.

3. Dialogautomaten

Die von einem Programm zu realisierenden Dialogfunktionen umfassen eine Information des Benutzers und die Benutzeraktionen. Für eine übersichtliche Spezifikation dieser Aufgaben genügt es, unter den Zuständen eines MEALY-Automaten Dialogzustände auszuzeichnen und in diesen Zuständen die Dialogfunktionen gesondert anzugeben /1/.

Definition 2: Ein Dialogautomat A_D ist ein Tupel

$$A_D = (X, Y, Z, f, g, D, i, f_D, g_D) \text{ mit}$$

X, Y, Z Mengen der Eingabezeichen, Ausgabezeichen und Zustände,
$D \subseteq Z$ Dialogzustände,

f	Abbildung aus $(Z \setminus D) \times X$ in Z	dialogfreie Überführung,
g	Abbildung aus $(Z \setminus D) \times X$ in Y	Ergebnisfunktion,
i	Abbildung aus D in Y	Benutzerinformation,
f_D	Abbildung aus $D \times X$ in Z	Benutzerorder,
g_D	Abbildung aus $D \times X$ in Y	Benutzeranfrage.

Ein Dialogautomat A_D definiert folgenden MEALY-Automaten

$$A = (X', Y', Z', f', g')$$

mit

$$X' = X,\ Y' = Y,\ Z' = Z,$$

und für $z \in Z'$ und $x \in X'$ ist

$$g'(z,x) = \begin{cases} g(z,x) \text{ für } z \in Z \setminus D, \\ g_D(z,x) \circ i(x) \text{ für } z \in D, \end{cases}$$

$$f'(z,x) = \begin{cases} f(z,x) \text{ für } z \in Z \setminus D, \\ f_D(z,x) \text{ für } z \in D. \end{cases}$$

Damit ist ein Dialogautomat als eine dialoggerechte Spezifikation eines MEALY-Automaten aufzufassen.

4. Dialogautomaten im Entwurfsprozeß

Beim Entwurf von Programmen mit Hilfe der I-Automaten sind die unter Verwendung von Dialogautomaten spezifizierten Dialogfunktionen durch Selektoren, Konstruktoren und Verarbeitungsproze-

duren zu konzipieren.
Ein Dialogautomat

$$A_D = (X, Y, Z, f, g, D, i, f_D, g_D)$$

sei gegeben. Die Informationsfunktion i wählt im Speicher (Zustand $z \in Z$) enthaltene Informationen aus, um dem Benutzer die gezielte Weiterarbeit zu ermöglichen. Sie entspricht damit der Kombination von Selektoren. Für einen Zustand $z \in D$ und eine Dialogeingabe x definieren $f_D(z,x)$ und $g_D(z,x)$ einen Arbeitsakt eines MEALY-Automaten und unterliegen damit den üblichen Prinzipien des Programmentwurfs.
Für den Entwurfsprozeß, bei dem das Problem der Erweiterung eines bereits entworfenen I-Automaten zu klären ist, ergeben sich folgende Entwurfsschritte:

1. Analyse der für den Dialog zusätzlich zu speichernden Informationen und Erweiterung des globalen Speichers;
2. Konzeption der zur Realisierung von f_D und g_D erforderlichen zusätzlichen Verarbeitungsprozeduren und Erweiterung des lokalen Speichers;
3. Konzeption der zusätzlichen Selektoren und Konstruktoren.

Da die Erweiterung eines I-Automaten durch nicht dialogspezifische Aufgaben in natürlicher Weise in einem Entwurfsprozeß möglich ist, fügt sich die Konzeption von Dialogfunktionen organisch in den gesamten Entwurfsprozeß ein. An dem angegebenen Modell des Syntaxanalyseteils eines Compilers wird deutlich, daß die Dialogmöglichkeiten, die von der vorhandenen Information ausgehen, unmittelbar aus dem Modell abzulesen sind. Den Benutzer interessierende Informationen sind:

gelesenes Zeichen,
Typangaben über Symbole.

Sinnvolle Eingriffe in den Programmablauf sind:

Überlesen von Zeichen,
Abbruch der Syntaxanalyse.

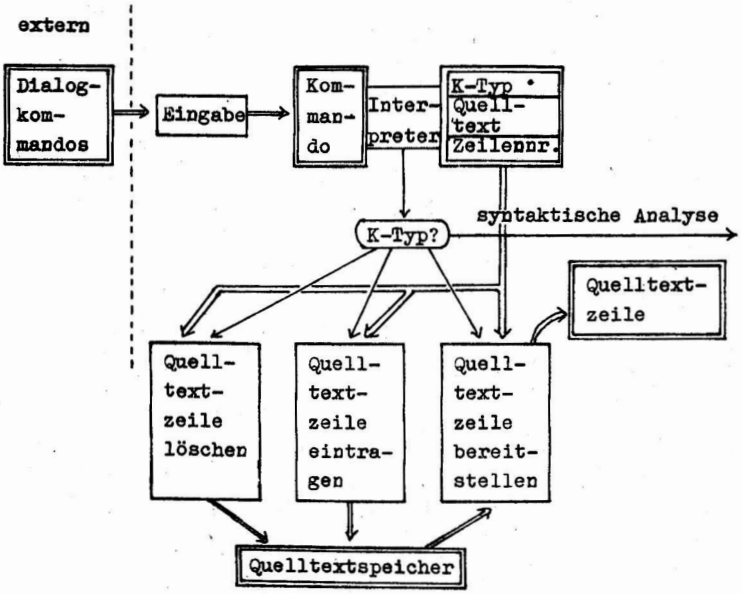
Ein Beispiel für eine Dialogaufgabe, bei der die gespeicherten Informationen zu erweitern sind, ist die zeilenweise Korrektur

von Quelltexten. An diesem Beispiel sollen die drei genannten Entwurfsschritte demonstriert werden.

1. Entwurfsschritt:
Zusätzlich ist der bereits verarbeitete Quelltext zu speichern. Die als nächste zu verarbeitende Quelltextzeile ist gesondert zu speichern, da die Zeichen der Zeile einzeln vom Syntaxanalyseprogramm angefordert werden.

2. Entwurfsschritt:
Eine Dialogeingabe heiße Kommando. Diese Kommandos sind zu interpretieren, d. h., die im Kommando enthaltene Information ist der vorgesehenen Verarbeitung zuzuführen.

3. Entwurfsschritt:
Ein Selektor ist für die Eingabe der Kommandos vorzusehen. Konstruktoren werden für die Verwaltung des Quelltextes benötigt (speichern, löschen, Zeile bereitstellen).



Modell für die Quelltextverwaltung im Dialog

Das obige Modell zeigt im besonderen, wie weitgehend unabhängig Dialogfunktionen konzipiert werden können, wenn es um die Verwaltung dialogspezifischer Informationen geht. Diese Erfahrungen wurden bei dem Entwurf und der Implementation eines Dialog-Entwurf- und Testsystems weitestgehend bestätigt.

Literatur

/1/ Hohberg, B.: Probleme der Dialogcompilation - Arbeitsweise und Realisierung des MS-Algol-Dialogcompilers. Diss. B, Humboldt-Universität, Berlin 1978

/2/ Hohberg, B., Kleffe, P., Kosciolowicz, R., Paulin, G., und Grafik, W.:
Mehrfachzugriffsystem MS - Anwenderbeschreibung MS-Algol. ZfR-Informationen ZfR-C-76. 10, Berlin 1976

/3/ Gluschkow, W. M.: Theorie der abstrakten Automaten. Berlin 1963

/4/ Deremer, F. L., and Kron, H. H.:
Programming-in-the-large versus programming in-the-small. Informatik-Fachberichte 1, 4. Fachtagung der GI, pp. 80 - 89, Berlin 1976

/5/ Bothe, K.: Spezifiaktion und Verifikation abstrakter Datentypen. ZfR-Informationen ZfR-78. 11, Berlin 1978

eingegangen: 07. 11. 1979

Anschrift des Verfassers:

Dr. sc. nat. Bodo Hohberg
Humboldt-Universität zu Berlin
Sektion Mathematik
Bereich Informationsverarbeitung
DDR-1080 Berlin
Unter den Linden 6

Rostock. Math. Kolloq. 14, 17 - 24 (1980)

Hardwin Jungclaussen

Sprachliche Aspekte des Entwurfs von Dialogsystemen und speziell von rechnerunterstützten Lehrsystemen

Die Nutzerfreundlichkeit eines Dialogsystems hängt wesentlich von seiner sprachlichen Ausstattung ab. Ihrem Entwurf gehen einige Festlegungen und Entscheidungen voraus, von denen drei besprochen werden sollen:

- die Dekomposition des Zielprozesses,
- die Wahl der Dialogprozeßklasse und
- die Wahl der Nutzersprachklasse.

1. Dekomposition des Zielprozesses

Der Prozeß, den der Nutzer gemeinsam mit dem Rechner im Dialog durchführt, soll Zielprozeß genannt werden. Ein solcher kann z. B. ein Konstruktions- oder ein Übungsprozeß sein. Die Kooperation zwischen Mensch und Rechner setzt eine Dekomposition des Zielprozesses in einzelne Operationen voraus. Eine Operation wird von einem Operator ausgeführt. Im folgenden wird der Operatorbegriff in einem erweiterten Sinn verwendet.

Definition 1: Ein Operator ist ein Gerät oder eine Vorschrift, das/die einem oder mehreren Operanden ein Resultat zuordnet. Wird die Zuordnung durch ein Gerät vorgenommen, so heißt der Operator real; wird sie durch eine Vorschrift festgelegt, so heißt er formal.
Mit op wird ein Operator, mit Op eine Menge von Operatoren bezeichnet.

Wesentliches Ziel der Dekomposition ist die Aussonderung der effektiv implementierbaren Operatoren. Ergebnis der Dekomposition ist die Menge Op_c der zu implementierenden Computeroperatoren op_c.

2. Wahl der Dialogprozeßklasse

Eine Analyse bestehender Dialogsysteme legt es nahe, drei kooperative (oder Dialog-) Prozeßklassen zu unterscheiden. Wir nennen sie:

- fixierten Prozeß mit Nutzereingriff,
- freien oder nichtspezialisierten Wechselprozeß,
- halbfreien oder spezialisierten Wechselprozeß.

Die erste Klasse ist dadurch charakterisiert, daß der gesamte Zielprozeß praktisch vollständig von vornherein durch ein Programm festgelegt ist. Dieses kann vom Nutzer entweder aus einer op_c-Bibliothek abgerufen oder von ihm selbst programmiert und eingegeben werden. Im zweiten Fall heißt es Nutzerprogramm. Der Nutzer hat die Möglichkeit, durch Eingriffe die Programmabarbeitung zu überwachen und evtl. Korrekturen anzubringen. In diese Klasse gehören die Systeme der interaktiven Programmaufbereitung und -abarbeitung. Ebenfalls sind die in /1/ und /2/ vorgestellten Systeme hier einzuordnen. Für die Formulierung der Nutzerprogramme wird in /1/ eine algorithmische, in /2/ eine Kommandosprache verwendet.

Die zweite Klasse ist dadurch charakterisiert, daß der Zielprozeß ein echter kooperativer Prozeß ist, dessen Gestaltung voll in der Hand des Nutzers liegt, der nach eigenem Gutdünken die Dienste des Rechners in Anspruch nimmt, indem er z. B. einen op_c aufruft, der die anstehende Operation ausführt, oder indem er einen op_c programmiert (z. B. in APL), eingibt und abarbeiten läßt. Dabei sind die einzelnen op_c in den Zielprozeß eingebettet und über diesen evtl. stark miteinander verkoppelt, z. B. durch die Übergabe von Operanden. Wenn die Verkopplung der op_c dem Nutzer während der kooperativen Arbeit aufgebürdet wird, so kann das System seinen Wert als Dialogpartner evtl. weitgehend verlieren.
Zur Überwindung dieser für freie Wechselprozesse charakteristischen Schwierigkeit haben sich zwei Wege als gangbar erwiesen. Der erste Weg wurde in /3/ und /4/ beschritten. Er besteht in der Einführung von Relationen in Op_c, welche die Verkopplungsmöglichkeiten zwischen den op_c angeben. Auf der Grundlage die-

ser dem Rechner bekannten Relationen ist es möglich, in jeder aktuellen Situation sowohl die Wahl des nächsten abzuarbeitenden op_c als auch dessen Einbindung in den Zielprozeß ganz oder teilweise zu automatisieren.

Der zweite Weg ist weniger elegant, jedoch zweckmäßig, wenn sehr viele Verkopplungsvarianten zwischen den op_c möglich sind und wenn die Menge Op_c leicht erweiterbar sein soll. Dieser Weg wird in /5/ und /6/ beschritten. Er führt zu der Klasse der halbfreien oder spezialisierten Wechselprozesse. Diese Klasse ist dadurch charakterisiert, daß eine neue Schicht, die Schicht der Dialogprogramme, und eine neue Sprache, die wir Autorensprache nennen wollen, in die Systemarchitektur eingefügt wird. Die Bezeichnung Autorensprache ist der Terminologie des rechnerunterstützten Unterrichts entlehnt.
Die Autorensprache dient der Formulierung von Dialogprogrammen. Der Autor eines Dialogprogramms spezialisiert durch dessen Implementierung das System für die dialogmäßige Lösung einer bestimmten Aufgabenklasse. Dabei hat er die Möglichkeit, vorhandene op_c zu verwenden, neue op_c zu programmieren und verschiedene Einbindungsvarianten der op_c in den Zielprozeß bereitzustellen. Er führt also Funktionen aus, die beim Beschreiten des ersten Weges vom System übernommen werden.
Ein implementiertes Dialogprogramm ist ein realer Operator. Seine Operanden sind Eingaben des Nutzers; mit anderen Worten, die Eingaben, die der Nutzer des spezialisierten Systems in der Nutzersprache formuliert, stellen Eingabedaten des Dialogprogramms dar.

3. Wahl der Nutzersprachklasse

Zur übersichtlichen Beschreibung der Sprachproblematik werden folgende Begriffe eingeführt:

Definition 2: Eine Struktur $(Op;K,L)$ heißt Operatorennetz; dabei ist $K = \{(op_i, op_j) : \text{Resultat von } op_i \text{ ist Operand von } op_j\}$,

$L = \{(op_i, op_j) : op_i \text{ wird vor } op_j \text{ abgearbeitet}\}$.

K und L heißen Kopplungs- bzw. Laufrelation.

Ein Operatorennetz kann seinerseits als Operator aufgefaßt werden, formal:

$$(\{bop\};\ K,L) = kop,$$

mit bop - Bausteinoperator und kop - kompositer Operator.

Durch Beziehungen der Art

$$kop^{(n)} = (\{bop\};\ K,L)^{(n)} = bop^{(n+1)}$$

können Schichtenarchitekturen aufgebaut werden. Der obere Index bezeichnet die Schicht. Handelt es sich dabei um formale Operatoren, so muß in jeder Schicht eine Sprache existieren, in der die Synthese kompositer Operatoren als Vorschrift angebbar ist.

Der kop-Synthese ist die kop-Rückführung entgegengesetzt. Die Rückführung eines formalen kop beinhaltet die Übersetzung der Synthesevorschrift in die Sprache einer niederen Schicht. Ein formaler Operator (bop oder kop) heißt Programm, wenn seine Rückführung bis auf die Schicht der Hardwareoperatoren eines Rechners gewährleistet ist.

Zur kop-Synthese gehören:

- Auswahl und evtl. Spezifizierung der bop; dabei ist unter Spezifizierung die Fixierung der variablen Parameter eines Operators zu verstehen;
- Festlegung der kop-Struktur;
- Wertzuweisung an die Operanden.

Die kop-Strukturierung umfaßt die Festlegung der Relationen K und L. Die Festlegung der Kopplungsrelation formaler kop erfolgt im allgemeinen dadurch, daß den Operanden Namen zugewiesen werden, die sowohl auf linken als auch auf rechten Seiten von Ergibtanweisungen auftreten können. Für die Festlegung der Laufrelation sind drei Vorgehensweisen denkbar:

- implizite Festlegung durch Operatoreigenschaften,
- explizite Festlegung durch Angabe von Laufvorschriften,
- explizite Festlegung durch Beschreibung von Laufvorschriften.

Ausgehend von der Sprachpraxis formalisierter Fachgebiete erscheint es zweckmäßig, zwei Arten der L-Festlegung zu definieren.

Definition 3: Die Festlegung einer Laufrelation heißt formelhaft, wenn sie implizit durch Prioritäts- und Assoziativitätseigenschaften der bop und explizit durch Klammerung erfolgt. Sie heißt algorithmisch, wenn sie formelhaft und/oder durch weitere Laufvorschriften erfolgt.

Eine Formel bzw. ein Algorithmus ist ein kop mit formelhafter bzw. algorithmischer L-Festlegung.
Nun lassen sich drei Sprachklassen definieren:

Definition 4: Eine Sprache heißt Kommandosprache bzw. Formelsprache bzw. algorithmische Sprache, wenn sie die op-Festlegung nur durch Spezifikation bzw. durch Synthese nur als Formel bzw. durch Synthese als Algorithmus ermöglicht.

Die Wahl der Nutzersprachklasse hängt vom Dialogziel ab. Häufig werden in der Nutzersprache ein kommandosprachlicher Teil und ein fachspezifischer algorithmischer oder formelsprachlicher Teil zusammengefaßt.

4. Komplettierung

Die Festlegung eines Operators von der Synthese als kop bis zur Arbeitsbereitschaft soll Komplettierung genannt werden. Grundsätzlich setzt das kooperative Arbeitsprinzip voraus, daß durch jede Nutzereingabe ein op_c komplett gemacht wird, so daß er abgearbeitet werden kann und dabei eine Ausgabe liefert.

Komplettierungen werden sowohl vom Nutzer als auch vom System vorgenommen. Komplettierungen durch den Nutzer können vom System angefordert werden. Die Nutzerfreundlichkeit verlangt, daß Komplettierungsarbeiten so weit wie möglich vom System ausgeführt werden. Dem dienen zwei wichtige Maßnahmen, die sich erheblich auf die Sprachkonzeption auswirken:

1. Minimierung des vom Nutzer aktiv zu beherrschenden Sprachschatzes z. B. durch das Arbeiten mit Formularen und Angeboten, die der Nutzer nur ausfüllt.
2. Automatische Erweiterung des aktuellen Sprachschatzes. Damit ist folgendes gemeint: Der Nutzer hat im Dialog die Möglichkeit, neue kop zu definieren, neue Namen für Operatoren

und Operanden einzuführen u.a.m. Es sollte ihm gestattet sein, auf früher verwendete Operatoren wieder zurückzugreifen oder einmal eingeführte Namen später wieder zu verwenden. Das bedeutet eine Erweiterung des Sprachschatzes, der dem Nutzer zur Verfügung steht. Um die Erweiterung sollte er sich nach Möglichkeit nicht kümmern müssen.

Die Abbildung 1 zeigt in einer vereinfachten Dialogsystemstruktur die Informationswege, die der Komplettierung dienen, und zwar im Falle eines spezialisierten Wechselprozesses. Über die Pfeile 1 bzw. 2 bzw. 3 werden übersetzte Nutzerprogramme oder Programmteile bzw. Bausteinprogramme (bop) bzw. Dialogprogramme übergeben. Über 4, 5 werden Dialogprogramme aufgerufen und über 6 op-weise dem Komplettierer zugeführt. Die Komplettierung des Dialogprogrammteils erfolgt über 7 durch Einbinden der erforderlichen bop, über 8 durch Spezifikation und über 9, 10 durch Operandenbereitstellung.

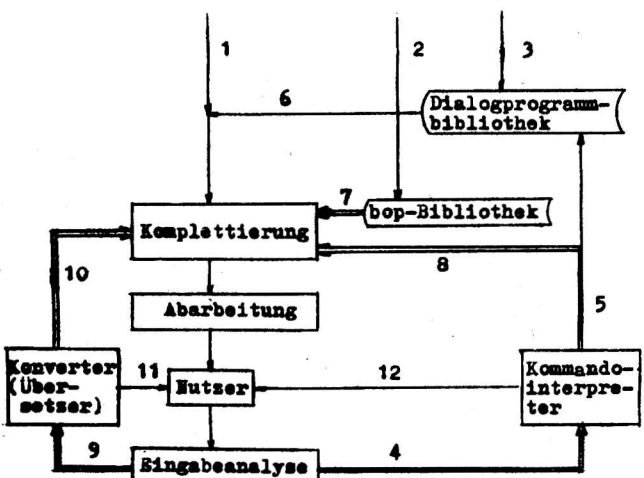


Abb. 1: Komplettierungsfluß.
Doppelpfeile kennzeichnen den Weg von Komplettierungsinformationen. Erläuterungen im Text.

Eine Besonderheit von Dialogsystemen, die der rechnerunterstützten Durchführung von Übungen in mathematisch-technischen Fächern dienen, besteht darin, daß über 9 beliebige Ergebnisse oder Zwischenergebnisse eines Aufgabenlösungsprozesses eingegeben werden können, also auch Formeln oder Programme, deren Richtigkeit vom System zu kontrollieren ist. Die Kontrolle übernimmt das Dialogprogramm, dem also als Operanden Operatoren angeboten werden können. Soll ein solcher Operator abgearbeitet werden, soll beispielsweise der Wert eines vom Nutzer (Studenten) eingegebenen booleschen Ausdrucks für eine bestimmte Wertebelegung der Variablen berechnet werden, so muß der Ausdruck übersetzt werden. Das kann zu einer unangenehmen Verflechtung der Rückführungs- und Komplettierungsprozesse führen. Im Sinne einer übersichtlichen Systemstruktur sollten beide Prozesse nach Möglichkeit entflochten werden. Das wird besonders wichtig, wenn das System zwei /6/ oder mehr Dialogprogrammschichten umfaßt.

Über die Pfeile 11 und 12 in der Abbildung werden dem Nutzer syntaktische Eingabefehler mitgeteilt.

Literatur

/1/ Dienbold, V., Heim, E., und Lunze, G.:
Ein dialogfähiges, modulares Simulationssystem für diskontinuierliche Prozesse unter Verwendung von Kleinrechnersystemen mit peripherem Zusatzspeicher. Ingenieurhochschule Dresden, Wissenschaftliche Beiträge 4, S. 2 (1978)

/2/ Frenzel, F. M., und Heltzig, H. F.:
Zur Steuerung selbstorganisierender Programmsysteme. Problemseminar "Rationalisierung der Projektierung automatisierter Informationssysteme", TU Dresden, Sektion Informationsverarbeitung, Geising 14. - 18. 11. 1977

/3/ Hegewald, E.: Ein Ansatz für die Problemklassenstrukturierung im Dialog. TU Dresden, Sektion Informationsverarbeitung (in Vorbereitung).

/4/ Kutschke, K.-H.: Ein einheitlicher Zugang bei der Modellierung von Dialogsystemen. Rostock. Math. Kolloq. 14, 25 - 46 (1980)

/5/ Monjau, D., und Ortleb, R.: Dokumentation DGA GD'71-KRS 4201-ESER, Seite Kleinrechner. Sektion Mathematik, Wiss.-Bereich MKR, Technische Universität, Dresden 1979

/6/ Piehler, H.: Problemorientierte Formulierung von Übungsprogrammen des rechnerunterstützten Hochschulunterrichts in mathematisch-technischen Lehrgebieten. Dissertation A, Technische Universität, Dresden 1977

/7/ Piehler, H., und Uhlemann, M.: Entwicklung und Nutzung problemorientiert formulierter Lehrprogramme für den rechnerunterstützten Hochschulunterricht. Wiss. Z. Techn. Univ. Dresden , 28 (1979), H. 1, S. 125

eingegangen: 07. 11. 1979

Anschrift des Verfassers:

Dozent Dr.-Ing. habil. Hardwin Jungclaussen
Technische Universität Dresden
Sektion Informationsverarbeitung
DDR-8027 Dresden
Momsenstraße 13

Rostock. Math. Kolloq. 14, 25 - 46 (1980)

Karl-Heinz Kutschke

Ein einheitlicher Zugang zur Modellierung von Dialogsystemen

1. Ziel und Einflußfaktoren einer Modellierung

Ein Modell sollte eine hinreichend gute Widerspiegelung eines bestimmten Teils der objektiven Realität sein. In unserem Fall betrachten wir Modelle für Dialogsysteme, ihre Herleitung und formale Beschreibung.
Eine allgemeine Aufgabe ist es, mit Hilfe mathematischer Modellierung einen Beitrag zur Erhöhung der Qualität von Dialogsystemen zu leisten, den Entwurf und die Implementierung zu unterstützen und die Anwendungen auf unterschiedliche Problemklassen zu erleichtern. Je nach spezieller Zielstellung werden sich auch die Modelle unterscheiden. Jedoch hängt die Modellierung nicht allein, wenn auch wesentlich, von der Zielsetzung ab. Weitere Faktoren sind: die mathematische Ausbildung und das Denkschema der Bearbeiter; die Art der auftretenden Probleme bei Entwurf, Implementierung und Anwendung; die Sicht des Bearbeiters auf das System.
Auf Dialogsysteme bezogen, lassen sich folgende Ziele anstreben:

- Exakte Beschreibung des Dialogsystems oder seiner Teile im Sinne einer konzeptuellen Ebene mit der Möglichkeit zur Ableitung der tatsächlichen Realisierung (interne Ebene) und der exakten Beschreibung der Nutzungsmöglichkeiten (externe Ebene) für die Arbeit mit dem System. Die Analogie zum Drei-Ebenen-Konzept der Datenbanktechnologie (vgl. z. B. /13/ und /14/) wurde bewußt gewählt.
- Nutzung mathematischer Modelle zur Gewinnung von praxisrelevanten Aussagen über bestimmte Eigenschaften des Systems und zum Aufbau von Algorithmen zur Prüfung, Korrektur und Rechnerunterstützung bei der Implementierung.

- Nutzung mathematischer Modelle als allgemeine Entwurfsgrundlage von Dialogsystemen (mathematische Entwurfstechnologie).
- Entwicklung einer mathematischen Theorie der Dialogsysteme (vergleichbar mit der Theorie formaler Sprachen für den Kompilerbau).

Der gegenwärtige Stand der Modellierung orientiert sich hauptsächlich an der ersten Zielsetzung, obwohl für die zweite bereits auf Einzelgebieten gute Ergebnisse vorliegen, z. B. in der Datenflußanalyse /2/ und in der Theorie der Programmschemata /6/. Andererseits ist eine exakte formale Beschreibung erste Voraussetzung für Realisierung, Algorithmisierung, Aussagengewinnung und Theorienbildung, insbesondere für eine rechnerunterstützte Entwurfs- und Implementierungstechnologie von Dialogsystemen. Bei der Wahl geeigneter Beschreibungsformen sind zwei Tendenzen erkennbar:

- Darstellung von Dialogsystemen bzw. ihrer Teile durch bereits bekannte mathematische Modelle. Der Vorteil liegt im Vorhandensein bereits ausgearbeiteter Theorien bzw. einer Basis für theoretische Untersuchungen. Verwendet werden Mengen-, Graphen-, Netz- und Automatentheorie, die Theorie formaler Sprachen, Programmschemata u. a. m.
- Erarbeitung von problemspezifischen Beschreibungshilfsmitteln, z. B. VDL /16/, Petri-Netze /11/. Dabei werden häufig für verschiedene Teile des Systems unterschiedliche Hilfsmittel benutzt /8/.

Wir wollen im wesentlichen der ersten Tendenz folgen, gehen prinzipiell vom Problemlösungsprozeß aus und wählen einen bestimmten strukturellen Aufbau der Dialogsysteme. Zur Begründung sei festgestellt, daß Dialogsysteme Hilfsmittel der Problemlösung sind. Die zugrundegelegte Struktur von Dialogsystemen dient der Orientierung für die Abbildung eines Problemlösungsprozesses in die Struktur des Modells eines Dialogsystems. Demnach sind zwei Voraussetzungen für die Modellbildung und Realisierung nötig:

- Kenntnisse des Problemlösungsprozesses,
- Kenntnisse der Grundstruktur des Dialogsystems.

Aus der Kenntnis der Prinzipien des Problemlösungsprozesses lassen sich grundlegende Modelle für Programmsysteme ableiten, die für die Modellierung von Dialogsystemen einen einheitlichen Zugang liefern können. Dabei reicht es zuerst aus, die Betrachtungen auf die drei Bestandteile eines Programmsystems:
- Daten (Datenbasis),
- Programmoduln (Programmbasis),
- Steuerung

zu orientieren (vgl. /12/).

Daten oder ein Datenbestand, der in einer Datenbasis gespeichert ist, sind eine Menge von Datenelementen. Die Menge aller Datenelemente des Systems bezeichnen wir mit D. Ein Datenelement fassen wir als Name-Wert-Paar auf. Einem Datenelement ist eineindeutig ein Name zugeordnet, wobei die Menge aller Namen von D mit N(D) bezeichnet wird. N(D) heißt auch Menge der Variablen. Jedem Element aus N(D) kann ein Wert aus einer hier nicht näher bestimmten Wertmenge zugeordnet werden. Die Modellierung von Datenbeständen zu einem Problemlösungsprozeß ist ein selbständiger Forschungsgegenstand und soll hier nicht in Betracht kommen.
Ein Programmodul ist ein abgeschlossenes Arbeitsprogramm zur Lösung einer wohldefinierten Teilaufgabe. Dabei bearbeitet jeder Modul Daten aus D.
Die Steuerung wollen wir einfach als Mechanismus zum Aufruf von Programmoduln auffassen.

2. Grundmodell eines Programmsystems

Aus Dekomposition und Komposition eines Problemlösungsprozesses ergibt sich ein allgemeines Grundmodell eines Programmsystems. Es besteht aus einer Menge von Programmoduln $P = \{P_1, P_2, \ldots, P_n\}$ mit einer Menge binärer Relationen $R = \{R_1, R_2, \ldots, R_m\}$ über P und kann zusammengefaßt werden als Menge von orientierten

Graphen $G = \{G_j : G_j = (P,R_j) \wedge R_j \in R\}$. Jedes $p_i \in P$ ist eine Abbildung

$$p_i : T_i \rightarrow T_{k(i)}$$

mit T_i, $T_{k(i)} \in \mathcal{P}(N(D))$. $\mathcal{P}(N(D))$ ist die Potenzmenge aller Namen der Datenmenge D. Wir sagen, p_i bearbeitet die Daten von $T_i \cup T_{k(i)}$. Die T_i heißen Variable der Eingabedaten und $T_{k(i)}$ Variable der Ausgabedaten von p_i. P liefert n Relationen über N(D) oder eine Relation V über $\mathcal{P}(N(D))$ mit

$$V = \{(T_i,T_{k(i)}) : T_i,T_{k(i)} \in \mathcal{P}(N(D)) \wedge p_i \in P \wedge p_i(T_i) = T_{k(i)}\}.$$

Aus V kann bei gegebenem Datenbestand D eine verfahrensorientierte Struktur für eine zugriffsoptimale Speicherung der Daten in der Datenbasis abgeleitet werden, indem unter Beachtung der Häufigkeit der Benutzung der Teilmengen von D entsprechende Zusammenfassungen errechnet werden. Einen Zugang dazu liefert das Konstituententheorem für Datenbestände /4/.

Wichtige Spezifizierungen des Grundmodells ergeben sich durch Festlegung der Relationen $R_i \in R$. Dazu gehört die zentrale Relation S. $(p_i,p_j) \in S$ gilt genau dann, wenn nach der Arbeit von p_i die Abarbeitung von p_j möglich ist, d. h., wenn zwei Lösungsschritte aufeinander folgen können. Der Graph $G_s = (P,S)$ wird Lösungsnetz oder Lösungsszenarium genannt. Jede Bahn von einem Anfangspunkt zu einem Endpunkt von G_s stellt einen Lösungsweg oder eine Lösungstrasse dar. Damit kann bereits das Hauptproblem der Arbeit mit einem Programmsystem, nämlich für eine gegebene Aufgabenstellung eine Lösungstrasse in G_s aufzubauen, beschrieben werden. Es ist erkennbar, daß dazu bestimmte Bedingungen erfüllt werden müssen. Wichtige von diesen basieren auf Datenabhängigkeiten.

Datenabhängigkeiten spezifizieren weitere Relationen des Grundmodells. Drei wichtige Datenabhängigkeiten sind die folgenden Relationen:

- Berührungsrelation B,
- Eingaberelation E,
- Ausgaberelation O.

Zwei Programmoduln (p_i,p_j) stehen in Relation B, wenn $T_{k(i)} \cap T_j \neq \emptyset$, in Relation E, wenn $T_i \cap T_j \neq \emptyset$, und in Relation O, wenn $T_{k(i)} \cap T_{k(j)} \neq \emptyset$ gilt. Auf die Relationen E und O ist bei der Realisierung besonders unter dem Aspekt der Datensicherung und Bedienbarkeit bei einer Parallelarbeit zu achten. Der Graph $G_B = (P,B)$ bestimmt wesentliche Architektureigenschaften und kann Grundlage einer Datenflußanalyse oder eines Modells der Ermittlung der Zuverlässigkeit sowie der Verifikation des Programmsystems sein. Zwischen G_B und G_s gibt es offensichtlich "bestimmte" Beziehungen. Über G_B läßt sich ein Netz von Prädikaten mit dem Individuenbereich D legen, das für die formale Beschreibung der Steuerung und der Semantik verwendet werden kann /1/. Nachfolgend erläutern wir ein spezielles Modell für die Steuerung, das vom Grundmodell ausgeht.

3. Modellierung der Steuerung

Gesteuert werden sollen Elemente aus P, und zwar unter Beachtung, daß über P orientierte Graphen existieren, die bei der Steuerung zu berücksichtigen sind. Der Einfachheit halber beschränken wir uns auf eine Relation N (z. B. N=S), Nachbarschaftsrelation genannt. Wir betrachten Teilgraphen $G^i = (P^i,N^i)$ von $G_N = (P,N)$, die wir Steuereinheiten nennen; i sei dabei Element einer geeigneten Indexmenge I, und allgemein gelte $P = \bigcup_{i \in I} P^i$ sowie $N_i \subseteq N$. Die Bildung der Steuereinheiten kann nach unterschiedlichen Kriterien erfolgen, z. B. durch Zusammenfassung von Moduln zu Teillösungskomplexen oder nach einer bestimmten Schichtung im Graphen G_N (vgl. /12/) oder G_S. Jeder Steuereinheit G^i ordnen wir ein sogenanntes Steuerprogramm P^i_{St} zu (vgl. Bild 1).

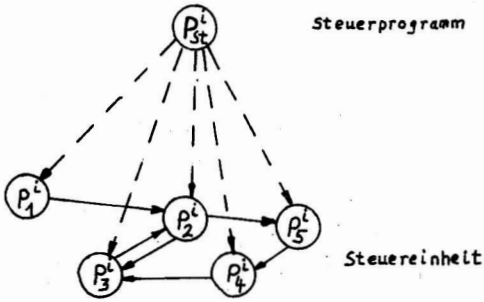


Bild 1: Steuereinheit mit Steuerprogramm

Rechentechnisch interpretieren wir P^i_{St} als ein Programm, das die Moduln P^i unter Beachtung von N^i aufruft. Der Prozeß der Bildung von weiteren Steuereinheiten und Steuerprogrammen wird dahingehend verallgemeinert, daß erzeugte Steuerprogramme zur Menge P hinzugefügt werden und mit diesen unter Hinzunahme von Elementen einer Nachbarschaftsrelation neue Steuereinheiten gebildet werden (siehe Bild 2).

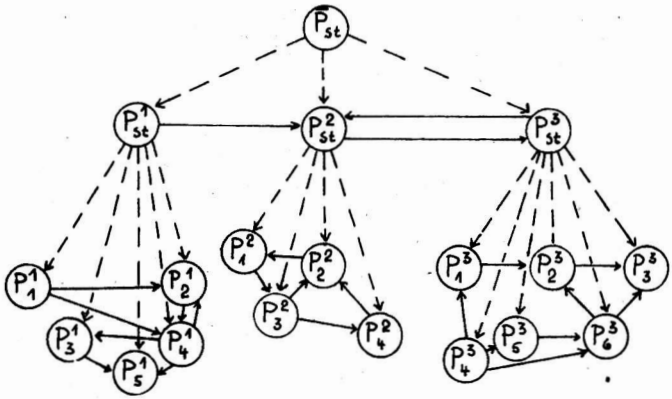


Bild 2: Einfache Hierarchie von Steuereinheiten

Diese rekursive Erweiterung bzw. Bildung von Steuereinheiten und Steuerprogrammen liefert über der Menge P der Arbeitsmoduln, erweitert um Steuerprogramme, neben der Nachbarschaftsrelation N (alte Bezeichnung, auch nach der Erweiterung beibehalten) eine weitere Ralation A, die Aufrufrelation. Es gilt $(p', p'') \in A$ genau dann, wenn p' Steuerprogramm von p" ist, d. h., wenn p" zu einer Steuereinheit gehört, deren Steuerprogramm p' ist, oder wenn p' p" aufruft. Der Graph $G_A = (P,A)$ liefert beispielsweise eine Grundlage für Segmentierungsalgorithmen (vgl. /7/ und /10/). G_A gibt nur an, welches Programm von welchem aufgerufen oder gesteuert wird. Für die Durchführung der Steuerung werden G_A und die Teilgraphen (Steuereinheiten) von G_N benötigt. Eine weitere Spezifizierung der Art und Weise der Steuerung kann durch die Einführung endlicher abstrakter Automaten erreicht werden.

Wir betrachten eine Steuereinheit von Arbeitsmoduln $G^i = (P^i, N^i)$ und fassen G^i als Zustandsdiagramm eines endlichen Automaten ohne Ausgabe auf. Dazu bilden wir P^i eineindeutig auf eine Zustandsmenge Z^i und N^i auf U^i, einer Menge von Zustandspaaren, ab, so daß G^i und $V^i = (Z^i, U^i)$ isomorphe Graphen sind, und führen eine Bewertung $B^i : U^i \rightarrow E^i$ ein. E^i fassen wir als Eingabealphabet auf und erhalten den bewerteten Graphen $u^i = (Z^i, U^i; B^i)$ als Zustandsdiagramm eines endlichen (partiellen) Automaten $A^i = (E^i, Z^i, u^i, s^i, \sum^i)$. s^i sei ein Anfangszustand und $\sum^i$ eine Menge von Endzuständen. Damit können wir die Arbeit jeder Steuereinheit als endlichen Automaten beschreiben. Die Sprache des Automaten A^i ist als Kommandosprache für die entsprechende Steuereinheit interpretierbar. Die rechentechnische Realisierung von A^i ist einfach und kann für jeden beliebigen Automaten A^i durch ein und dasselbe Programm mit unterschiedlichen Zustandstafeln geliefert werden.

Jedem Automaten A^i können wir jetzt einen Zustand aus einer Zustandsmenge zuordnen, neue Automaten aufbauen und zu Automatenhierarchien (Automatenschichten) gelangen. Die Automatenhierarchie beschreibt die Steuerung und liefert die Kommando-

sprache eines Programmsystems. Jedem Automaten A_i entspricht bei der vorgenommenen Konstruktion eineindeutig ein Zustand, dem eine Zustandstafel zugeordnet ist. Durch Interpretation eines jeden Zustands, d. h. des entsprechenden Zustandswortes, ist eine rechentechnische Realisierung sehr transparent. Das Zustandswort muß Auskunft geben, ob ein Zustand einem Programmmodul oder einer Zustandstafel zugeordnet ist. Auf Einzelheiten müssen wir hier verzichten, obwohl aus diesem Modell weitere rechentechnische Vorteile gezogen werden können (z. B. Fehlerbehandlung, Austauschbarkeit, Überprüfung der Erreichbarkeit).

Für eine sequentielle Abarbeitung reicht das Modell des endlichen Automaten aus. Sind jedoch nebenläufige, bedingungsabhängige oder parallel arbeitende Programme zu steuern, empfiehlt sich der Übergang zu Petri-Netzen, /15/. Formal ist der Modellbildungsprozeß völlig analog zum vorangegangenen mit Automaten. Anstelle eines Automaten A^1 ist ein geeignetes Petri-Netz einzusetzen.
Die Wirksamkeit des Modells der Automatenhierarchie soll nachfolgend auf den Entwurf der Steuerung bzw. des Kommunikationsteils von Dialogsystemen angewandt werden.

4. Abstrakte Automaten und Entwurf eines Dialogsystems

Der Ausgangspunkt zu den folgenden Betrachtungen sei einmal das Grundmodell nach Abschnitt 2 und zum anderen die Struktur für Dialogsysteme mit ihren 5 Hauptbestandteilen Datenbasis, Verarbeitungsmoduln, Kommunikations-, Abfrage- und Graphikteil (vgl. Bild 3, /5/).

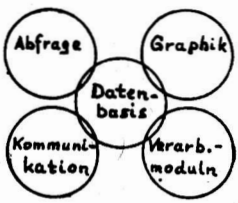


Bild 3: Hauptbestandteile eines Dialogsystems

Wir beschränken uns hier jedoch auf ein Grundmodell G = (P,N) mit nur einer Relation N und betrachten lediglich Verarbeitungsmoduln P mit dem Kommunikationsteil D, wie in Bild 4 dargestellt.

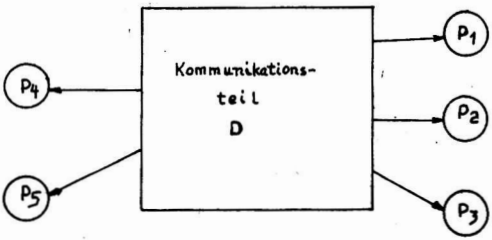


Bild 4: Kommunikationsteil mit Verarbeitungsmoduln

Dabei gehen wir davon aus, daß der Aufruf von Moduln aus P über den Kommunikationsteil realisiert wird.
Das Ziel der weiteren Betrachtungen besteht darin, den Kommunikationsteil schrittweise zu verfeinern, so daß N und solche Bedingungen berücksichtigt werden, die die konkreten Kommunikationsmöglichkeiten (Hard- und Software) bieten. Unter Verwendung des Modells eines abstrakten Automaten gelangen wir von Bild 4 direkt zu einem Zustandsdiagramm (Bild 5) eines endlichen Automaten, des Aufrufautomaten AM.

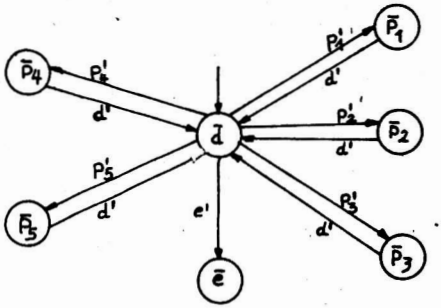


Bild 5: Zustandsdiagramm eines Aufrufautomaten

Der Übergang kann wie folgt beschrieben werden.
Jedem Programmodul $p_i \in P$ ordnen wir einen Zustand $\overline{p}_i$ zu und dem Kommunikationsteil D den Zustand $\overline{d}$. Führen wir weiter einen speziellen Zustand, den Endzustand $\overline{e}$ ein, so erhalten wir eine Zustandsmenge $ZM = \{\overline{p}_1, \overline{p}_2, \ldots, \overline{p}_n, \overline{d}, \overline{e}\}$. Mit dem Eingabealphabet $EM = \{p'_1, p'_2, \ldots, p'_n, d', e'\}$ kann für das Strukturschema nach Bild 4 ein Zustandsdiagramm in Bild 5 angegeben werden. Dabei wurden nur die Kanten eingezeichnet, die zur Sprache von AM gehören und die für die Steuerung relevante Übergänge darstellen. Durch Hinzufügen eines zusätzlichen Zustandes läßt sich aus dem angegebenen partiellen Automaten ein entsprechender vollständiger Automat angeben. Dieser hinzugefügte Zustand kann beispielsweise einem Fehlerprogramm zugeordnet sein.
Verzichten wir auf die Ausgabe, so läßt sich der Aufrufautomat formal durch $AM = (EM, ZM, u, \overline{d}, \{\overline{e}\})$ mit der Zustandsüberführungsfunktion u angeben. Eine programmtechnische Realisierung des Automaten AM kann durch ein zentrales Steuerprogramm ZSP und des Zustandes $\overline{d}$ durch einen Analysator A erfolgen (man vgl. /3/). Unter diesem Aspekt läßt sich ein entsprechender Graph der Aufrufe mit ZSP als Wurzel in Bild 6 darstellen.

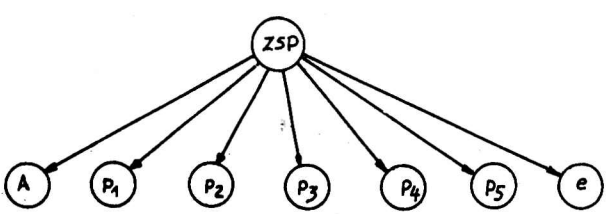


Bild 6: Graph der Programmaufrufe (A Analysator)

Der Graph der Aufrufe zeigt nur an (durch angegebene Bögen), daß entsprechende Programme aufgerufen werden. Das Zustandsdiagramm und damit der Automat AM gibt die Art und Weise des Aufrufs (Übergangs) an. Dabei wird folgende Interpretation gewählt: Dem Übergang von einem Zustand s zum Zustand s^+ entspricht der Aufruf des zugeordneten Programms p^+, wenn die Arbeit des Programms p beendet und das nötige Eingabesignal vorliegt. Dabei entspricht s dem p und s^+ dem p^+. Die spezielle Struktur des Zustandsdiagramms von AM hängt von der gewählten Grundstruktur des Dialogsystems ab. Tabelle 1 gibt die Zustandstafel von AM als Matrix (ZM,EM) wieder und zerfällt in bestimmte Teilmatrizen.

Mit dem Automaten AM wurde eine erste Schicht der Steuerung beschrieben und eine vollständige Basis für die Realisierung der Steuerung der Aufrufe geliefert.

Im Zustand $\bar{d}$ erfolgen die Eingaben durch den Nutzer. Im Modell AM wird aber nur die Eingabe von Zeichen $\alpha \in EM$ ausgewiesen. Als Aufgabe steht außer der Bereitstellung der Mitteilung, welcher Modul aufgerufen wird, auch die Bereitstellung von Parametern für den jeweiligen Verarbeitungsmodul. Dazu ist die prinzipielle Arbeitsweise des Wechsels zwischen Arbeit des Analysators und einem Verarbeitungsprogrammodul zu fixieren.

Es ergibt sich somit als nächste Stufe der Modellverbesserung eine Verfeinerung des Kommunikationsteils zur Kommunikationsstruktur.

Wir betrachten die verarbeitungsmodulspezifische Arbeit des Analysators und gehen davon aus, daß für jeden Modul spezielle Parameter bereitgestellt werden. Dazu ordnen wir jedem Verarbeitungsmodul eine dem Modul angepaßte, sogenannte K (ommunikations)-Einrichtung zu (vgl. /9/). Eine K-Einrichtung faßt z. B. die Menüs, die einem Teilproblem zugeordnet sind, und weitere Hilfsmittel der Kommunikation zusammen. Wir erhalten für Paare (PK-Paare genannt) von Verarbeitungsmoduln p (P-Einrichtungen in /9/) und K-Einrichtungen k ein entsprechendes Zustandsdiagramm (Bild 7).

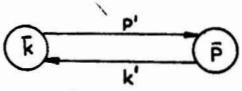


Bild 7: Zustandsdiagramm eines PK-Paares

Dabei ordnen wir jeder K-Einrichtung k einen Zustand $\bar{k}$ zu und wählen p'als Eingabesignal für den Übergang von $\bar{k}$ nach $\bar{p}$ und analog k' als Eingabesignal für den Übergang von p nach $\bar{k}$. Führen wir diesen Prozeß des Aufbaus von PK-Zustandspaaren für alle $p \in P$ aus, so erhalten wir eine Menge von Zustandspaaren (vgl. Bild 8), die zu einem sogenannten Zustandsdiagramm der Kommunikationsstruktur zu verbinden sind. Beim Aufbau des Zustandsdiagramms für die Kommunikationsstruktur sind bestimmte Bedingungen zu beachten. Eine wesentliche Bedingung ist die Verbindung von Teillösungen (Insellösungen) im Sinne des Aufbaus eines Problemlösungsszenariums und damit die Relation N des Grundmodells.

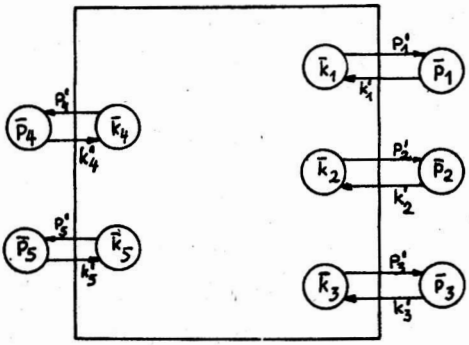


Bild 8: Menge von PK-Zustandspaaren

Wir verbinden nicht die Verarbeitungsmoduln, sondern die entsprechenden K-Einrichtungen. In zweiter Linie sind Hard- und Softwareeinschränkungen sowie eine zweckmäßige Strukturierung

der Kommunikationsstruktur zu beachten. Die Berücksichtigung derartiger Bedingungen führt zur Hinzunahme von weiteren K-Einrichtungen mit reinen Verteilerfunktionen. Wir werden die entsprechenden Zustände mit $\overline{k}_{zi}$ (i Element einer Indexmenge) bezeichnen und wählen einen Zustand davon als Anfangszustand $\overline{k}_o$ aus.

Mit der Menge $\overline{ZV} = \{\overline{k}_o, \overline{k}_1, \ldots, \overline{k}_n, \overline{k}_{z1}, \ldots, \overline{k}_{zr}\}$ von Zuständen, die den K-Einrichtungen zugeordnet sind, der Menge $\overline{EV} = \{k'_o, k'_1, \ldots, k'_n, k'_{z1}, \ldots, k'_{zr}\}$ von Eingabesignalen und einer partiellen Überführungsfunktion $\overline{uk}$ können wir einen K-Verbindungsautomaten $\overline{AV} = (\overline{EV}, \overline{ZV}, \overline{uv}, k_o, \emptyset)$ formal notieren.

Die Übergangsfunktion (partielle) $\overline{uv} : \overline{EV} \times \overline{ZV} \rightarrow \overline{ZV}$ liefert alle Übergänge zwischen den Kommunikations-Einrichtungen.

Der abstrakte Automat $\overline{AV}$ ist anstelle des Zustandes $\overline{d}$ im Automaten AM zu setzen. Dabei ist die Kopplung mit der Umgebung von $\overline{d}$ sinnvoll vorzunehmen. Zur Erläuterung dieser Kopplung sei auf die Zustandstafel (Tabelle 1) verwiesen, aus der sich auch eine allgemeine Vorgehensweise ergibt. Prinzipiell ist in Tabelle 1 eine Anzahl von Substitutionen vorzunehmen, und wir erhalten Tabelle 2. Im Ergebnis dieser eindeutig beschreibbaren Verfeinerung kann Tabelle 2 als Zustandstafel eines sogenannten Verbindungsautomaten AV angegeben werden. Bild 9 zeigt ein analoges Zustandsdiagramm.

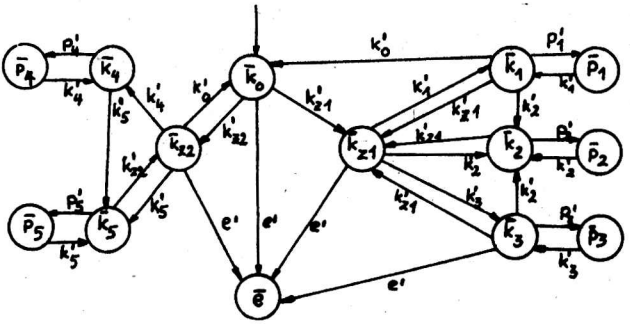


Bild 9: Zustandsdiagramm eines Verbindungsautomaten

Formal kann als Ergebnis der oben erwähnten Substitutionen notiert werden

$AV = (\overline{EV}, ZV, uv, \overline{k}_0, \{\overline{e}\})$ mit

$\overline{EV} = \overline{EV} \cup EM$,

$ZV = \overline{ZV} \cup ZM$ und uv als partielle Übertragungsfunktion entsprechend Tabelle 2.

Programmtechnisch realisiert der Analysator (als Programmodul dem $\overline{d}$ zugeordnet) den abstrakten Automaten $\overline{AV}$ und das zentrale Steuerprogramm über zwei Stufen den abstrakten Automaten AV. Eine weitere Stufe der Verfeinerung bedingt die Berücksichtigung von Dialogeingaben. Das Hilfsmittel dazu sind Eingabeautomaten für einzelne Datentypen. Jeder Verarbeitungsmodul benötigt eine Liste von Eingabeparametern, die die K-Einrichtung bereitzustellen hat (vgl. /9/). Jeder Parameter ist ein Datenelement eines bestimmten Typs.

Gegeben sei eine Menge T von Datentypen $T = \{t_1, t_2, \ldots, t_r\}$.

Ein Verarbeitungsmodul p habe die Eingabeparameterliste $(l_1, l_2, \ldots, l_m)$. In einer PASCAL-ähnlichen Schreibweise kann man eine Vereinbarung wie folgt festlegen

$$\underline{\text{modul}}\ p(l_1 : t_{i1}, l_2 : t_{i2}, \ldots, l_m : t_{im}).$$

Wir nehmen zur Demonstration des Sachverhalts die folgenden Datentypen als gegeben an, deren Eingabe soft- bzw. hardwaremäßig abgesichert sei. Tatsächlich greifen wir hier nur auf t_1, t_2 zurück.

t_1 Skalar von Typ REAL
t_2 beliebig lange Folge von Daten des Typs t_1
t_3 Skalar von Typ INTEGER
t_4 Folge von Daten des Typs t_3
t_5 Bezeichnung (Namen)
t_6 Folge von Bezeichnungen
t_7 Koordinatenpaar
t_8 Folge von Koordinatenpaaren

u.s.f.

Für die Eingabe eines Datenelements vom Typ $t_i \in T$ kann jeweils ein Eingabeautomat definiert werden, z. B. für t_1 E_{11}

mit dem in Bild 10 dargestellten Zustandsdiagramm und der Zustandstafel in Tabelle 3.

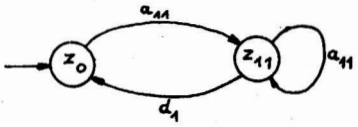


Bild 10: Zustandsdiagramm eines Eingabeautomaten E_{11}

Häufig werden die Parameter eines Moduls so vereinbart, daß Parameter eines Typs unmittelbar aufeinanderfolgen. Dann ergibt sich z. B. ein Automat E_{13} für drei Parameter des Typs t_1 nach Bild 11.

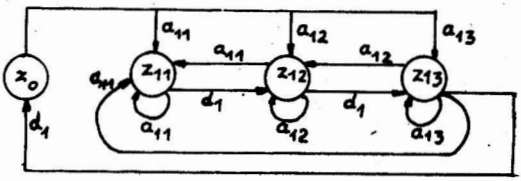


Bild 11: Zustandsdiagramm eines Eingabeautomaten E_{13}

Eine Zustandstafel für E_{14} stellt Tabelle 4 dar. Die spezielle Struktur dieser Automaten ist wesentlich bedingt durch

- einfache Möglichkeiten der Erzeugung von E_{1i} aus E_{1i-1},
- die Art, wie die Eingabe beim zugrundegelegten Terminalsystem erfolgen kann.

Für den Übergang von E_{1i-1} auf E_i ist in der Zustandstafel die Veränderung des ersten Elements der letzten Zeile (vgl. Tabelle 4) und die Hinzunahme einer Zeile und Spalte mit uniformer Angabe der Elemente nötig.
Zur Interpretation der Arbeit eines Automaten E_{1i} gelte: Im Zustand z_{1i} kann der i-te Parameter der Teilliste von Paramerern

des Typs t_1 eingegeben werden, d_1 stellt das vollständig eingegebene und zulässige Eingabewort (Zahl) dar. Das Eingabesignal a_{1i} erzwingt den Zustand z_{1i}. Gegebenenfalls ist zur Eingabe von d_1 eine weitere Verfeinerung nötig.
Ein Datenelement des Typs t_2 kann dem im Bild 12 dargestellten Zustandsdiagramm eines Eingabeautomaten E_{21} zugeordnet werden. Bild 12 enthält außerdem das Zustandsdiagramm eines Automaten E_{22} für die Eingabe von zwei Datenelementen des Typs t_2. Die Tabellen 5 und 6 lassen auch hier die Bildung der Automaten E_{2i} aus E_{2i-1} ($i > 1$) in sehr einfacher Weise erkennen.

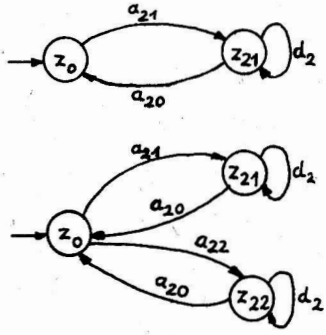


Bild 12: Zustandsdiagramme für Eingabeautomaten E_{21} und E_{22}

Da im allgemeinen Parameter unterschiedlichen Typs für einen Modul auftreten, ergibt sich die Aufgabe, Eingabeautomaten E_{ji} mit unterschiedlichen Typen t_j und Vielfachheiten i zu koppeln. Eine formale Beschreibung und Realisierung ist einfach. Auf Einzelheiten wollen wir hier verzichten. Die programmtechnische Realisierung erfolgt so, daß einem Automaten E_{ji} ein Unterprogramm UP_j innerhalb des Analysators entspricht.
Für jede Parameterliste, die mittels einer K-Einrichtung aufzubauen ist, läßt sich ein spezieller Eingabeautomat angeben. Jeder Eingabeautomat ist aus elementaren Automaten E_{j1} kombi-

nierbar. Die nächste Aufgabe ist die Substitution von Zuständen $\bar{k}$ durch Eingabeautomaten bei richtiger Einbettung in die Umgebung von $\bar{k}$. Den durch diesen Verfeinerungsprozeß entstehenden (partiellen) Automaten wollen wir Kommunikationsautomaten nennen.

Eine Einbettung von Eingabeautomaten kann wie folgt vorgenommen werden.
Wir bezeichnen einen Eingabeautomaten, der für den Zustand $\bar{k}$ eine vollständige Parameterliste realisiert, mit AE_k. Ein Eingabeautomat AE_k habe den Anfangszustand z_o, den wir bei der Substitution von $\bar{k}$ durch AE_k mit $\bar{k}$ identifizieren. Die Menge der Zustände von AE_k besteht aus m Zuständen außer z_o, wenn in der Parameterliste des entsprechenden Moduls m Parameter vereinbart wurden. Bei der Substitution von $\bar{k}$ durch AE_k legen wir für die Einbettung folgendes fest, dabei beziehen wir uns auf das entsprechende Zustandsdiagramm:

- Alle Vorgänger von $\bar{k}$ bleiben erhalten (z_o wurde mit $\bar{k}$ identifiziert).
- Alle Nachfolger von $\bar{k}$ bleiben ebenfalls Nachfolger von $\bar{k}$ und werden auch Nachfolger mit den gleichen Bogenbewertungen von denjenigen Zuständen z_{1r} in AE_k, die einfachen Datentypen entsprechen. In unserem Beispiel sind es Datentypen aus $T_e = \{t_1, t_3, t_5, t_7\}$, d. h. die Zustände z_{1r} mit $1 = 1,3,5,7$.

Die hier getroffene Festlegung für die Einbettung entspricht den praktischen Erfahrungen bei der Realisierung eines digitalgraphischen Dialogsystems. Andere Einbettungsbedingungen sind möglich. Der Hauptgrund dafür, daß diejenigen Zustände z_{ir}, denen Datenfolgentypen entsprechen, nicht mit der Umgebung verbunden werden, liegt in der Bereitstellung zusätzlicher Hilfsmittel (Hilfsmenüs) für den Nutzer. Hier kann eine weitere Verfeinerung dieser Zustände zur Präzisierung erfolgen. Die gewählten Einbettungsbedingungen gestatten außerdem eine einfache Handhabung des Zustandsdiagramms (bzw. Zustandstafel), da die Substitution nur eng umrissene Teilmatrizen erfaßt, also nur lokalen Charakter hat. Dieser Sachverhalt ermöglicht auch die zweckmäßige Ausnutzung der Hierarchie bzw. Schichtung der end-

lichen Automaten, die durch den Verfeinerungsprozeß geliefert wird. Durch die lokale Einbettung ergeben sich im Zustansdiagramm Teilmatrizen, deren Elemente nicht definiert sind. Der hierarchische Aufbau und die hierarchische Arbeitsweise bei der Steuerung untersützt auch die Prüfung der richtigen Kommandos in Abhängigkeit vom jeweiligen Zustand sehr effektiv, da die Prüfung nur lokal durchgeführt wird. Jeder eingebettete Automat, unabhängig von der Hierarchiestufe, in die er eingesetzt wurde, besitzt seine eigene, exakt angebbare Kommandoteilsprache.

Die dargelegten Ausführungen zeigen die Beschreibung der Steuerung eines Dialogsystems durch abstrakte Automaten in drei Ebenen:

- Aufrufautomat,
- Verbindungsautomat,
- Kommunikationsautomat.

Jeder Ebene entspricht eine bestimmte Steuerstruktur. Jedes Absteigen in eine höhere Steuerstruktur bedeutet eine Übergabe der Steuerung an einen anderen Automaten, der für einen Zustand "eingesetzt" wird. Jedes Aufsteigen von einer Struktur zu einer niedrigeren bedeutet ein Verlassen des einen Automaten und Fortsetzen der Arbeit in der übergeordneten Steuerstruktur. In der programmtechnischen Realisierung entsprechen die Zustände der einzelnen Automaten folgender Zuordnung:

Aufrufautomat: Verarbeitungsmoduln und Analysator.
Verbindungsautomat: Kommunikationshilfsmittel, wie Menüs und weitere Daten.
Eingabeautomaten: Unterprogramme des Analysators. Jedem Datentyp entspricht ein Unterprogramm mit entsprechenden Parametern.

Damit ist ein schichtweiser, klar strukturierter Aufbau und eine exakte Beschreibung der Steuerung gegeben; Veränderungen in einer Schicht beeinflussen die Steuerung in anderen Schichten nicht. Die Automaten, speziell deren Zustandsdiagramme, geben die Kommandosprache exakter wieder als jede verbale Beschreibung. Aus der Beschreibung der Struktur durch Automaten wird auch ein Zugang zur Entwiklung einer Programmiersprache

für die Beschreibung der Steuerung möglich und eine Programmstruktur eindeutig ableitbar.

		d'	P' p'_1	p'_2	...	p'_n	e'
P	$\bar{p}_1$	$\bar{d}$					
	$\bar{p}_2$	$\bar{d}$					
	⋮	⋮		—			—
	$\bar{p}_n$	$\bar{d}$					
$\bar{d}$		-	$\bar{p}_1$	$\bar{p}_2$	...	$\bar{p}_n$	$\bar{e}$
$\bar{e}$		-	-	-	...	-	-

Tabelle 1: Zustandstafel eines (partiellen) Aufrufautomaten

		d' k'_1	k'_2	...	k'_n	k'_{z1} ... k'_{zr} k'_o	P' p'_1	p'_2	...	p'_n	e'
P	$\bar{p}_1$	$\bar{k}$	-		-						
	$\bar{p}_2$	-	k'_2		-	-		-			-
	⋮				⋮						
	$\bar{p}_n$	-	-	...	$\bar{k}_n$						
d	$\bar{k}_1$						$\bar{p}_1$	-	...	-	
	$\bar{k}_2$							$\bar{p}_2$	...	-	
	⋮							• • •			
	$\bar{k}_n$						-	-	...	$\bar{p}_n$	
			$\overline{uv}(k'_i, \bar{k}_j)$								$\overline{uv}(e', \bar{k}_j)$
	$\bar{k}_{z1}$										
	$\bar{k}_{z2}$							-			
	⋮										
	$\bar{k}_{zr}$										
	$\bar{k}_o$										
$\bar{e}$					-			-			-

Tabelle 2: Zustandstafel eines (partiellen) Verbindungsautomaten

	d_1	a_{11}
z_o	-	z_{11}
z_{11}	z_o	z_{11}

Tabelle 3: Zustandstafel des Eingabeautomaten E_{11}

	d_1	a_{11}	a_{12}	a_{13}	a_{14}
z_o	-	z_{11}	z_{12}	z_{13}	z_{14}
z_{11}	z_{12}	z_{11}	-	-	-
z_{12}	z_{13}	z_{11}	z_{12}	-	-
z_{13}	z_{14}	z_{11}	z_{12}	z_{13}	-
z_{14}	z_o	z_{11}	z_{12}	z_{13}	z_{14}

Tabelle 4: Zustandstafel des Eingabeautomaten E_{14}

	a_{20}	d_2	a_{21}
z_o	-	-	z_{21}
z_{21}	z_o	z_{21}	-

Tabelle 5: Zustandstafel des Eingabeautomaten E_{21}

	a_{20}	d_2	a_{21}	a_{22}	a_{23}
z_o	-	-	z_{21}	z_{22}	z_{23}
z_{21}	z_o	z_{21}	-	-	-
z_{22}	z_o	z_{22}	-	-	-
z_{23}	z_o	z_{23}	-	-	-

Tabelle 6: Zustandstafel des Eingebeautomaten E_{23}

Literatur

/1/ Bachmann, K.-H.: Automatensysteme und ihre Anwendung zur Definition von Programmierungssprachen. EIK 11, 214 - 220 (1975)

/2/ Bachmann, P.: Datenflußanalyse in schwach interpretierten Graphenschemata. Diss. (B), Fachbereich Mathematik-Kybernetik, Akademie der Wissenschaften der DDR, Berlin 1978

/3/ Berndt, E.: Realisierung eines Dialogkerns bei simulierter Dialog- Ein- und Ausgabe. Bericht WPU-DIGRA Nr. 49, Wilhelm-Pieck-Universität, Rostock 1979

/4/ Bessabotnikov, Ju. I., i Simonov, B. I.: K voprosu racional'noj organizacii informacionnych massivov na informacionnych poljach ASU. Algoritmy i organizacija rešenija ékonomiceskich zadač, Vypusk 4, Statistika, str. 17 - 31, Moskva 1974

/5/ Encarnacao, J.: Systemtechnologische Aspekte von CAD-Systemen. Informatik-Fachberichte 11, Methoden der Informatik für rechnerunterstütztes Entwerfen und Konstruieren, GI-Fachtagung. S. 20 - 51, München 1977

/6/ Ershov, A. P.: Theory of program schemata. Proceedings of the IFIP Congress 1971. Amsterdam 1971

/7/ Gendt, G.: Aufbau und Zerlegung von Programmsystemen in Verbindung mit dem DIGRA 73-System. Diss. (A), Wilhelm-Pieck-Universität, Rostock 1974

/8/ Kupka, I., Oberquelle, H., und Wilsing, N.: An Experimental Language for Conversational Use. Bericht 18, Institut für Informatik, Universität Hamburg 1975

/9/ Kutschke, K.-H.: Einheitliches Konzept der Realisierung der Kommunikation Mensch-EDVA in Dialogsystemen. Rostock. Math. Kolloq. 5, 41 - 57 (1977)

/10/ Pätz, T.: Eine Methode zur automatischen Erzeugung von Overlaystrukturen. Rostock. Math. Kolloq. 14, 65 - 81 (1980)

/11/ Petri, C. A.: Concepts of net theory. MFCS'73 Conf. Rec, 137 - 146, Bratislava 1973

/12/ Salter, K. G.: A Methodology for Decomposing System Requirements into Data Processing Requirements in Software Engineering. 2-nd international conference on Software Engineering, 13 - 15 October 1976, San Francisco, California, pp. 91 - 101

/13/ Schlageter, G., und Stucky, W.: Datenbanksysteme, Konzepte und Modelle. Stuttgart 1977

/14/ Schubert, D.: Stand und Perspektiven der Entwicklung von Datenbankbetriebssystemen. Rechentechnik/Datenverarbeitung 14, 4. Beiheft, 2 - 10 (1977)

/15/ Starke, P. H.: Petri-Netze. ZKI-Informationen Nr. 3/1979, Akademie der Wissenschaften der DDR, Zentralinstitut für Kybernetik und Informationsprozesse, Berlin 1979

/16/ Zemanek, H.: Abstrakte Objekte. Elektronische Rechenanlagen 10, H. 5, 208 - 217 (1968)

eingegangen: 07. 11. 1979

Anschrift des Verfassers:

Prof. Dr. sc. nat. K.-H. Kutschke
Wilhelm-Pieck-Universität Rostock
Rechenzentrum
DDR-2500 Rostock
Albert-Einstein-Straße 21

Rostock. Math. Kolloq. 14, 47 - 54 (1980)

Klaus Müller

Leistungsanalyse und Bewertung von Teilnehmersystemen

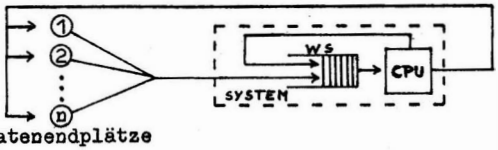
1. Einführung

Die dargestellten Ergebnisse basieren auf den Erfahrungen, die bei der Entwicklung und Nutzung des Teilnehmersystems FINDIPSY /1/, /2/ an der Technischen Hochschule Karl-Marx-Stadt gesammelt wurden. Das Teilnehmersystem FINDIPSY ist als experimentelles System entstanden, das nach seiner Fertigstellung einer Reihe von Untersuchungen unterzogen wurde. Es ist offensichtlich, daß die dabei gewonnenen Resultate im Rahmen dieser Arbeit nicht in ihrer Gesamtheit aufgezeigt werden können (vgl. dazu /3/, /4/). Wir beschränken uns daher auf Ergebnisse, die mit einem einfachen Warteschlangenmodell und mit einem Signalstrommodell /5/ erzielt worden sind.

2. Einfaches Warteschlangenmodell

Für die Untersuchungen wurde folgendes Modell zugrunde gelegt (vgl. /6/) (Bild 1):

- n gleichartige Nutzer schicken von n Datenendplätzen Anforderungen an das System.
- Diese Anforderungen werden nach dem Prinzip der zyklischen Zuordnung verarbeitet.
- Jeder Nutzer kann nur dann eine neue Anforderung an das System richten, wenn die vorangegangene zu Ende bearbeitet wurde.



Datenendplätze

Bild 1: Grundmodell eines Systems mit zyklischer Zuordnung und n Anforderungsströmen

Weiterhin gelten folgende Voraussetzungen:

(a) Die Antwortzeit des Systems soll gegenüber der Denkzeit der Nutzer (von Beendigung einer Anforderung bis zur Formulierung der nächsten) vernachlässigbar sein.

(b) Der zum Zeitpunkt t im System ankommende Anforderungsstrom sei ein POISSONscher Strom mit der Intensität

$$(n-k) \cdot \lambda,$$

wobei k die Anzahl der Nutzer repräsentiert, die zum Zeitpunkt t eine Anforderung im System in Bearbeitung haben.

(c) Die Bedienzeit der Anforderungen sei exponential verteilt mit der Verteilungsfunktion

$$B(t) = 1 - e^{-\mu t}.$$

Daraus lassen sich folgende Ergebnisse ableiten:

(a) Mittlere Länge E der Warteschlange WS:

$$E^{(n)} = \sum_{k=0}^{n} k \cdot \varrho^k \frac{n!}{(n-k)!} \frac{1}{\sum_{i=0}^{n} \varrho^i \cdot \frac{n!}{(n-i)!}},$$

wobei

$$\varrho = \frac{\lambda}{\mu} \text{ ist.}$$

(b) Antwortzeit T:

$$T = (E^{(n-1)}+1) \cdot t$$

mit t - erforderliche Bedienzeit.

(c) Relative Antwortzeit $\hat{T}$:

$$\hat{T} = \frac{T}{t} = E^{(n-1)} + 1$$

Das Verhalten der mittleren Warteschlangenlänge E in Abhängigkeit von ϱ und n ist aus Bild 2 ersichtlich.

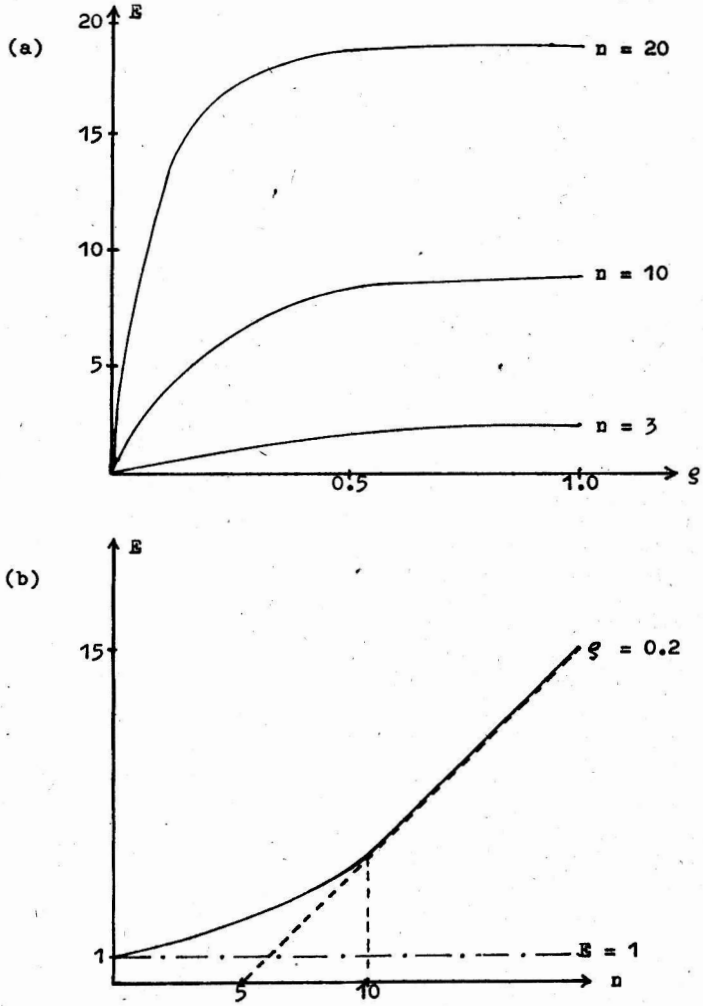


Bild 2: Verhalten der mittleren Warteschlangenlänge E in Abhängigkeit von ϱ und n.

Bei der Bewertung existierender bzw. Konzipierung neuer Systeme sind die Größen

n - Nutzeranzahl,

$\frac{1}{\mu}$ - mittlere Bedienzeit,

$\frac{1}{\lambda}$ - mittlere Denkzeit,

$\hat{T}$ - relative Antwortzeit

von besonderer Bedeutung.
Bei existierenden Systemen ist es möglich, den Parameter ϱ (Verhältnis von Bedienzeit zur Denkzeit) anhand üblicher Abrechnungsinformationen (Sitzungszeit und verbrauchte CPU-Zeit) abzuschätzen, indem man ϱ_I mit

$$\frac{1}{|I|} \cdot \sum_{i \in I} \frac{B_i}{S_i - B_i} \leq \varrho_I \leq \frac{1}{|I|} \sum_{i \in I} \frac{B_i}{S_i - \sum_{j \in I} B_j}$$

über hinreichend viele Sitzungsperioden mittelt.
Dabei ist B_i - CPU-Zeit des Nutzers i,

S_i - Sitzungszeit des Nutzers i,

I - Indexmenge der mit Nutzer i gleichzeitig arbeitenden Nutzer (Sitzungsperiode) und

ϱ_I - Verhältnis von Bedienzeit zur Denkzeit innerhalb der Sitzungsperiode I.

Mit der Kenntnis von ϱ ist aber die relative Antwortzeit nur noch von n abhängig. Damit ist die Frage nach einer optimalen Nutzeranzahl N berechtigt.
Zur Festlegung von N sind unterschiedliche Betrachtungsweisen möglich:

(a) $N^{(a)} = \frac{1}{\varrho}$.

Jeder der $N^{(a)}$ Nutzer sendet λ Aufträge pro Zeiteinheit an das System, das $\lambda \cdot N^{(a)}$ Anforderungen pro Zeiteinheit verarbeiten kann.

(b) $N^{(b)} = \frac{1}{\varsigma} + 1.$

Hierbei kennzeichnet $N^{(b)}$ das Verhältnis von Denkzeit und Bedienzeit zur Bedienzeit /5/.

(c) $N^{(c)} \approx \frac{2}{\varsigma}$.

In diesem Fall legt $n = N^{(c)}$ einen Punkt auf der Kurve $\hat{E}^{(n)}$ fest, von dem ab $\hat{E}$ linear wächst (vgl. Bild 2 (b)).

Bei der Konzipierung neuer Systeme lassen sich anhand der Vorstellungen zu Nutzeranzahl (n), Nutzerprofil (λ) und Antwortzeitverhalten ($\hat{T}$) Anforderungen an die Bedienzeiten ($\frac{1}{\mu}$) ableiten bzw. ausgehend von erreichbaren Bedienzeiten Schlußfolgerungen bez. Nutzeranzahl bzw. Antwortzeitverhalten ziehen. Aussagen über die Zusammensetzung der Bedienzeit - genauer: welche Komponenten des Systems beeinflussen die Bedienzeit am stärksten - sind über das Signalstrommodell möglich.

3. Signalstrommodell

Mit Hilfe dieses Modells /7/ ist es möglich, ausgehend von einem Anforderungsstrom $\eta_B = 1$ des Nutzers an das System, die Ströme durch die einzelnen Strukturen S_i des Systems zu ermitteln (Bild 3):

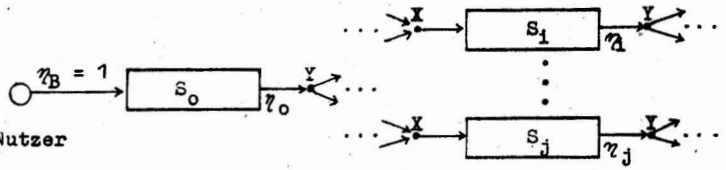


Bild 3: Signalströme durch die einzelnen Strukturen S_i

An den Stellen X addieren sich die eintreffenden Ströme und an den Stellen Y werden sie meßbaren Wahrscheinlichkeiten entsprechend aufgespalten. η_i bedeutet dann, daß die Struktur S_i bei

einer Anforderung des Nutzers im Mittel η_i mal aufgerufen wird. Hinter den einzelnen Strukturen verbergen sich bestimmte, für die Arbeit notwendige Hauptspeichermengen. Auf der Basis der berechneten η_i läßt sich ein Maß φ_i für die Nutzung dieser Bereiche definieren:

$$\varphi_i = \frac{\eta_i}{s_i}$$

(s_i - Anzahl der von S_i benötigten Bytes).

Damit sind Ansatzpunkte gegeben für die Entscheidung, welche Strukturen hauptspeicherresident und welche transient gemacht werden sollten.
Schließlich läßt sich auf der Basis der im Mittel abzuarbeitenden Anzahl b_i von Befehlen innerhalb der Strukturen ein Maß c_i für die mittlere CPU-Belastung durch die einzelnen Strukturen angeben:

$$c_i = \eta_i \cdot b_i.$$

Damit werden Aussagen gewonnen, welche Strukturen in welchem Maße Einfluß auf die Bedienzeit des Gesamtsystems bez. Nutzeranforderung haben.

4. Schlußbemerkungen

Die Anwendung der vorgestellten Modelle auf das Teilnehmersystem FINDIPSY lieferte folgende Resultate:

(a) Beobachtungen im Rechenzentrum der THK ergaben für das vorhandene Nutzerspektrum und eine EC 1040 mit BSAN-Bildschirmperipherie für den Parameter ϱ folgende gemittelte Abschätzung:

$$0{,}08 \leq \varrho \leq 0{,}2.$$

Damit folgt für die optimale Nutzeranzahl

$$5 \leq N^{(a)} \leq 12{,}5\ ,$$

$$6 \leq N^{(b)} \leq 13{,}5\ ;$$

$$10 \leq N^{(c)} \leq 25.$$

(b) Prozeß- und E/A-Steuerung erwiesen sich (wie erwartet) als die Engpässe unter den einzelnen Strukturen von FINDIPSY bez. der CPU-Zeit.

(c) Der mittlere Wert φ_K für die Speichernutzung durch die funktionellen Komponenten des Systems lag mit

$$\varphi_K = 0{,}062 \cdot 10^{-3}$$

relativ niedrig im Gegensatz zu Prozeß- und E/A-Steuerung mit

$$\varphi_P = 9{,}5 \cdot 10^{-3} \text{ bzw.}$$

$$\varphi_{EA} = 0{,}94 \cdot 10^{-3}.$$

Diese Tatsache gibt zumindest der Frage nach der Auslagerung bestimmter funktioneller Komponenten eine Berechtigung.

(d) Prozeß- und E/A-Steuerung haben einen ca. 30%igen Anteil an der CPU-Zeit der Systemprozesse gegenüber den Nutzerprozessen.

Weitere Ergebnisse vgl. /4/.

Literatur

/1/ Teilnehmersystem FINDIPSY - eine Einführung.
Nutzerinformation 1/79, Techn. Hochschule, Sektion Rechentechnik und Datenverarbeitung, Karl-Marx-Stadt 1979

/2/ Teilnehmersystem FINDIPSY - Anwenderhandbuch.
Techn. Hochschule, Sektion Rechentechnik und Datenverarbeitung, Karl-Marx-Stadt 1979

/3/ Teilnehmersystem FINDIPSY - Problemdokumentation.
Techn. Hochschule, Sektion Rechentechnik und Datenverarbeitung, Karl-Marx-Stadt 1979

/4/ Müller, K.: Modelluntersuchungen am Teilnehmersystem FINDIPSY. Bericht 2/79, Techn. Hochschule, Sektion Rechentechnik und Datenverarbeitung, Karl-Marx-Stadt 1979

/5/ Bergholz, G.: Zur Analyse der Multiprogrammbearbeitung in einer Prozeßrechenanlage. rechentechnik/datenverarbeitung, beiheft 3, 61 - 66 (1978)

- " - : Zur Ermittlung der Forderungsstromintensitäten in einem Echtzeitoperationssystem für Prozeßrechenanlagen. Wiss. Z. Techn. Univ. Dresden, 24, 3/4, 563 - 570 (1975)

/6/ Kleinrock, L.: Ausgewählte analytische Ergebnisse über Teilnehmerrechensystemen. In: Beihefte zur Zeitschrift "Elektronische Rechenanlagen" Teilnehmer-Rechensysteme, Vorträge zur Fachtagung der NTG in Erlangen vom 20. - 27. 09. 1967, S. 45 - 73, München und Wien 1968

/7/ Polze, C.: Strukturmessungen am Mehrfachzugriffssystem MS. AdW der DDR, ZfR-Informationen. ZfR-78.09, S. 28 - 61, Berlin 1979

eingegangen: 07. 11. 1979

Anschrift des Verfassers:

Dipl.-Math. Klaus Müller
Technische Hochschule Karl-Marx-Stadt
Sektion Rechentechnik und Datenverarbeitung
DDR-9001 Karl-Marx-Stadt
Straße der Nationen 62

Rostock. Math. Kolloq. 14, 55 - 63 (1980)

Rainer Ortleb

Zur Modellierung von Dialogprogrammen

1. Dialogsystem und Dialogprogramm

Dialogsysteme sind, von der gerätetechnischen Basis her beginnend, durch i. a. hierarchische Softwaresysteme in ihren Verhaltensweisen geprägt. Ein Dialogprogramm ist dabei ein Programm, das diese Verhaltensweisen (vorwiegend problemorientiert) weiter spezifiziert. Bei hierarchischer Softwarearchitektur des Dialogsystems ist in diesem Sinne jede aufsteigende Stufe Dialogprogramm des darunterliegenden abgeschlossenen Systems. Eine Modellierung von Dialogprogrammen dient damit sowohl der Abstraktion der Systementwicklung als auch der Entwicklung effektiver, u. U. auch rechnergestützter Technologien der Dialog-Anwendungsprogrammierung.

2. Funktionen

Über bestimmte, relativ wenig spezifizierte Funktionen läßt sich die grundlegende Verhaltensweise von Dialogpartnern beschreiben. Die Verfeinerung dieser Funktionen durch Anreicherung ihrer unterscheidbaren Informationsträger gestattet die Ableitung weiterer Eigenschaften. Nachfolgendes beschränkt sich auf die Ansätze.

2.1. Aktion, Intuition, Reaktion

Der Dialog wird zunächst von Seiten des Dialogpartners Mensch betrachtet. Eine Menge A möglicher Aktionen ist vorgegeben. Die i. a. nicht präzisierbare Intuition l wirkt als Auswahl einer Aktion $a \in A$ durch $a = l(A)$. Die Reaktion des anderen Dialogpartners ist die Zuordnung $r : A \to \Phi$, die als $\varphi = r(a) \in \Phi$ der Aktion a aus der Menge seiner Handlungen Φ die spezielle Handlung φ zuordnet. Mit $\varphi = r(l(A)) = (l \circ r)(A)$ wird diese letztendlich durch eine Auswahl aus A bestimmt.

Formal gleichberechtigt erscheinen die Dialogpartner, wenn man festlegt:

$$A_i := A,$$
$$A_j := \Phi,$$
$$r_{ij} : A_i \to A_j \text{ mit } a_j = r_{ij}(a_j) \text{ und } a_i = l_i(A_i),$$
$$r_{ji} : A_j \to A_i \text{ mit } a_i = r_{ji}(a_j) \text{ und } a_j = l_j(A_j).$$

Damit führt z. B. die Intuition l_i des Partners i zu einem Auswahlergebnis a_i aus A_i, das per Reaktion r_{ij} des j-ten Partners (bez. des Partners i) eine Handlung a_j zur Folge hat:

$$a_j = l_i(A_i),$$
$$a_j = r_{ij}(a_i) = (l_i \circ r_{ij})(A_i).$$

Interpretiert man die in der Form der Darstellung über allgemeine Partner i und j vollziehbare Vertauschung der Partner für das Paar Mensch-Maschine, dann erhält man eine 'maschinenorientierte' Betrachtungsweise, die etwa am Beispiel eines Spieles plausibel wird, bei dem die Maschine Handlungen produziert, auf die der Mensch durch Aktionen reagiert.

2.2. Parallele und serielle Zerlegung einer Funktion

Ist für zwei Mengen X und Y eine Funktion $\varphi : X \to Y$ gegeben und gilt für X_i (i=1,...,m) und Y_j (j=1,...,n)

mit $X = X_1 \times X_2 \times \ldots \times X_m$ bzw. $Y = Y_1 \times Y_2 \times \ldots \times Y_n$ die Beziehung

$$y = (y_1, y_2, \ldots, y_n) = (\varphi_1(x_1, x_2, \ldots, x_m), \ldots, \varphi_n(x_1, \ldots, x_m)) = \varphi(x)$$

bei $x \in X$,

$$x_i \in X_j \quad (i=1,\ldots,m)$$

und $y_j \in Y_j$ (j=1,...,n),

dann heiße $(\varphi_1, \ldots, \varphi_n) := \varphi$ eine parallele Zerlegung der Funktion φ.

Wird eine Funktion φ durch Hintereinanderausführung $\varphi = \varphi_1 \circ \varphi_2 \circ \ldots \circ \varphi_n$ durch $\varphi_j : Y_{j-1} \to Y_i$ mit j=1,...,n und $X = Y_0$, $Y = Y_n$ realisiert, wobei zwischen X, Y und Y_j

(j=1,...,n-1) nicht notwendig weitere Beziehungen bestehen sollen, dann werde hier von einer seriellen Zerlegung der Funktion gesprochen. Die Funktion φ als Handlung φ zu interpretieren, bedeutet, die Definitions- und Wertebereiche X und Y als für die Betrachtung als Handlung unwesentlich anzusehen. Das entspricht der Auffassung, Dialogprogramme in zwei Sprachebenen zu implementieren /1/.

2.3. Zeit- und Kopplungsverhalten von Funktionen

Zeit- und Kopplungsverhalten von Funktionen sind nichtrechentechnisch i. a. vernachlässigbare Eigenschaften. Hinsichtlich des Zeitverhaltens soll qualitativ und quantitativ unterschieden werden. Qualitativ gesehen ist eine Funktion wiederholend, wenn ihre Realisierung ohne äußeres Zutun (nach Ablauf von Zeiteinheiten) erneut beginnt, und nichtwiederholend, wenn dieser Realisierung ein erneuter Anstoß vorausgehen muß. Quantitativ sind damit Dauer und Frequenz einer Funktion begründet. Das Kopplungsverhalten bedarf immer dann keiner Beachtung, wenn keine Gleichzeitigkeit der Realisierung von Funktionen vorliegt. Wird eine auf eine Zeitskala bezogene Beschreibung vermieden, erweist sich das Führen von Ausführungs- und Erfüllungsbedingungen (geben Anfang frei bzw. zeigen Ende an) als zweckmäßig.

3. Einige Aspekte eines Dialogprogramm-Modells

3.1. Aktions-Reaktions-Graph

Wird die Zuordnung Reaktion $r : A \to \Phi$ mit $\varphi = r(a)$ als

$$(r,a) \xrightarrow{\lambda} \varphi$$

aufgefaßt und um die Möglichkeit des Übergangs zu einer neuen Zuordnung $\tilde{r}$ durch

$$(r,a) \xrightarrow{\delta} \tilde{r}$$

erweitert, so kann ein (endlicher) Automat $[A, \Phi, R, \lambda, \delta]$ unter Einhaltung entsprechender Bedingungen für die Mengen A, Φ und R dargestellt werden. Dabei sind r und $\tilde{r}$ Elemente der Reak-

tionsmenge R. Wird $[A, \Phi R, \lambda, \delta]$ als Graph dargestellt, so erhält man einen Aktions-Reaktions-Graph. Der Aktions-Reaktions-Graph sei als Aktions-Niveau-Graph (Dialogplan /3/) bezeichnet, falls die Betrachtung weniger dem problemunabhängigen Übergangsverhalten (Dialogreaktionen) als dem problemabhängigen Ergebnisverhalten (Problemreaktion, Handlung) gilt.

3.2. Aktionsworte

Aus Elementen der Aktionsmenge $A = \{a_1,\ldots,a_n\}$ wird ein Aktionswort $a_{i(1)}\ a_{i(2)} \cdots a_{i(m)}$ mit $i(j) \in \{1,\ldots,n\}$ und $j=1,\ldots,m$ zusammengestellt. Die einer Aktion durch Reaktion folgende Handlung eines Systems ist durch dieses bestimmt oder kann programmiert werden. Je nachdem, ob die Handlung systemgegeben oder programmierbar ist, werde die Aktion innere oder äußere Aktion genannt. Nach weiteren Eigenschaften der Handlungen φ heiße eine Aktion a außerdem

- Übergangsaktion, falls $\varphi + \varphi \neq \varphi$ *) ist,
- Nullaktion, falls $\varphi + \varphi = \varphi$ ist,
- leere Aktion, falls φ leere Handlung ist.

Je nach Art der zum Aktionswort zusammengestellten Aktionen sind aus diesem im Dialogsystem gleichwirkende Aktionsworte gemäß den nachfolgenden Vorschriften ableitbar
$(a,b,e,\underline{e},n,\underline{n},u,\underline{u} \in A)$:

(1) falls u innere Übergangsaktion, dann $u \rightarrow u$:

(2) falls $\varphi = r(\underline{u})$ mit $\varphi + \varphi \neq \varphi$ *), dann $\underline{u}$ äußere Übergangsaktion und $\underline{u} \rightarrow \underline{u}$:

(3) falls n innere Nullaktion, dann $na \ldots bn \rightarrow a \ldots bn$

(4) falls $\varphi = r(\underline{n})$ mit $\varphi + \varphi = \varphi$, dann $\underline{n}$ äußere Nullaktion und $\underline{n}a \ldots b\underline{n} \rightarrow a \ldots b\underline{n}$

(5) falls 1 innere leere Aktion, dann $al \rightarrow a$ bzw. $la \rightarrow a$

(6) falls $\varphi = r(\underline{1})$ leere Handlung, dann $\underline{1}$ äußere leere Aktion und $a\underline{1} \rightarrow a$ bzw. $\underline{1}a \rightarrow a$

Wird das Aktionswort nach (1) bis (6) behandelt, so entsteht ein Aktionswort, in dem jeder Doppelpunkt ein Wortteil ab-

*) Dabei wird unter $\varphi + \varphi$ eine zweimalige Ausführung der Handlung φ verstanden.

schließt, dem eine Handlungsfolge zugeordnet wird, die eine Handlung enthält, deren Wiederholung nicht zum gleichen Ergebnis führt. Da (2), (4) und (6) von φ bedingt werden, hängen sie von der Zuordnung r ab und damit schließlich auch von einem Anfangszustand im automatentheoretischen Sinne. (1), (3) und (5) gelten ebenso zustandsbezogen, werden aber hinsichtlich der Programmierung nicht mit Handlungen in Verbindung gebracht, so daß auch nicht von Handlungen sondern von Eigenschaften der Aktionen ausgegangen wird.

3.3. Datenform einer Aktion

Für die programmtechnische Realisierung eines Aktions-Reaktions-Graphen ist die Erzeugung einer Datenform der Aktion zweckmäßig, die zugleich als Aktionsmenge A' einer fiktiven Dialogeinrichtung verstanden werden kann. Von realer Dialogeinrichtung und deren Aktionsmenge A ausgehend, wird A' als $A' = \{a' | a' = d(a)\}$ über eine Vorschrift d erzeugt. Die Reaktion r wird als $r = d \circ r'$ im wesentlichen über eine auf A' bezogene Reaktion r' dargestellt.

4. Beispiel: Niveauorientierte Programmierung

Die sogenannte niveauorientierte Programmierung /2/ wurde zur Unterstützung der programmierenden Nutzer des digitalgeometrischen Arbeitsplatzes (DGA) der Konfiguration GD'71-KRS 4201 auf Basis FORTRAN entwickelt. Der Nutzer arbeitet dabei mit dem Aufruf von Niveau-Routinen, deren Eigenschaften die bequeme Realisierung von Dialogplänen gestatten.
Als Aktionsmenge wird beispielsweise für die Standards NIVØ, NIV1,...,NIV6 die Menge $A = \{a, v_1, \ldots, v_i, h, e_1, \ldots, e_j, i\}$ definiert (vereinfacht). Dabei ist unter a, v_k, h, e_l, i im einzelnen zu verstehen ($1 \leq k \leq i$, $1 \leq l \leq j$):

- a ... Tastendruck auf beliebige Taste der alphanumerischen Tastatur,
- v_k ... Drücken oder Lichtstiftidentifikation der k-ten Vordergrundtaste (VT),

h ... Drücken oder Lichtstiftidentifikation einer Hintergrundtaste (HT),

e_1 ... Lichtstiftidentifikation eines 1-ten Eingabetextes (ET),

i ... Lichtstiftidentifikation eines Objektes, das nicht VT, HT oder ET ist.

Den Bildschirmobjekten wird im Bestand des Programms der Charakter VT, HT, ET oder keiner davon zugewiesen.
Für z. B. NIV4 ist die Datenform A' der Aktionsmenge A durch die Menge aller (j + 4)-Tupel
(Name, x-Koordinate, y-Koordinate, Taste, $Eingabe_1$,..,$Eingabe_j$)
gegeben (j ... Anzahl der e_1 aus A). In der Programmierung spiegelt sich diese Datenform im Aufruf

```
CALL NIV4(NAME,X,Y,TASTE)
```

und den hier nicht näher erläuterten Zuweisungen bez. e_1 wider. Dabei lautet die Vorschrift zur Erzeugung der Datenform:

a → Übergabe der Positionierkugelkoordinaten auf X,Y,

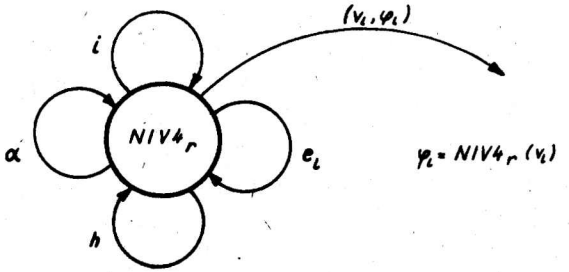
v_k → Übergabe der Tastennummer auf TASTE und Verlassen von NIV4,

h → Übergabe der Tastennummer auf TASTE,

e_1 → Übernehme einer dadurch angeforderten alphanumerischen Eingabe entsprechend der Zuweisung für e_1,

i → Übergabe des Objektnamens auf NAME.

In Form einer Graphendarstellung hat NIV4 also folgende Gestalt:



Ein Aktionswort ahh $e_3e_1av_3$iah z. B. geht wegen

a, e_1, h, i ... innere Nullaktionen,

v_k ... äußere Übergangsaktion

in das Wort ahhe$_3$e$_1$av$_3$:iah über, dem im betrachteten Zustand (Niveau) ahhe$_3$e$_1$av$_3$: bzw. he$_3$e$_1$av$_3$: entspricht.

Nach v_3 gelten die Aktionseigenschaften, die infolge des Übergangs eintreten. Der ab Doppelpunkt unverändert gebliebene Wortteil wird danach neu betrachtet. Untersuchungen der so akzeptierten Worte bzw. Wortteile erlauben Aussagen über Kommunikationsmöglichkeiten und Kommunikationssicherheit.
Als Beispiel aus der niveauorientierten Programmierung für weitere spezielle Aktionen nach (1) bis (6) können angegeben werden:

- für innere Übergangsaktion: Interrupttaste Bedienschreibmaschine;
- in der Floskel

```
1   CALL NIV4(NAME,X,Y,TASTE)
    IF (TASTE.EQ.5) R=SQRT(X*X+Y*Y)
    GOTO 1
```

  wird für $v_k = v_5$ eine äußere Nullaktion und für $v_k \neq v_5$ eine äußere leere Aktion ausgedrückt;
- für innere leere Aktion: Irrung einer Lichtstiftidentifikation durch ununterbrochenes Picken.

In der Bestimmung der Feinarchitektur der NIV-Routinen, die ihrerseits auf einen gemeinsamen Kern zurückgeführt werden, wird von serieller und paralleler Zerlegung von Funktionen Gebrauch gemacht.
Fragen des Zeitverhaltens spielen bei zyklisch organisierten Lese- und Schreibprozessen (Registerabfragen, Displayfilemanipulationen) eine Rolle. Kopplungseigenschaften sind bei der für den Anwender zugänglichen Programmierung von Unterbrechungsbehandlungen zu beachten. So sind z. B. beim skizzierten NIV4 während der Ausführung von Handlungen HT- und VT-Aktionen zugelassen, d. h., daß erst nach Abarbeitung der zur äußeren Übergangsaktion gehörenden Handlung die dem Übergang entsprechende neue Zuordnung r (Reaktion) wirksam wird.

5. Rechnerunterstützungen

Der angedeutete Formalismus hat sich beim Entwurf der niveauorientierten Programmierung als nützlich erwiesen. Man muß allerdings bedenken, daß die Übersichtlichkeit mit zunehmender Komplexität des real behandelten Falles mehr oder weniger verloren geht. Da die wesentlichen Funktionen jeweils durch Zuordnungen und nicht etwa durch analytische Ausdrücke gegeben sind, erscheint schon bei diesen Konstruktionsarbeiten wünschenswert, dazu den Rechner einzusetzen. Für die Stützung der dem dargestellten Modell entsprechenden niveauorientierten Programmierung wurden durch zwei, derzeit noch nicht abgeschlossene Projekte Ansätze gemacht:
Zum einen wurde ein Dialog-im-Dialog-System DIDSYS entworfen /4/, das mit Hilfe interaktiver Bildschirmarbeit entwickelte Dialogpläne unter Voraussetzung bereitgestellter Handlungs-Moduln rechenfähig umsetzt.
Zum anderen steht ein Programm GENE /2/ zur Verfügung, das codierte Aktionsmengen bzw. deren Datenform speicher- bzw. dateigerecht generiert. Ein diesem Generierungsprogramm entsprechendes Dialogsystem GEMA, das durch interaktive Arbeit die Codierung erübrigt, wurde daraus weiterentwickelt.

Literatur

/1/ Bauböck, E.: Dialogue-handling system for graphics and CAD applications. In: Proceedings International Conference Interactive Techniques in Computer Aided Design, ACM Italian Chapter, Universita di Bologna, S. 438 - 444, Bologna 1978

/2/ Ortleb, R.: Niveauorientierte Programmierung. Dokumentation DGA GD'71-KRS 4201-ESER, Sektion Mathematik, Wissenschaftsbereich Math. Kybernetik und Rechentechnik, TU Dresden, Dresden 1979

/3/ Ortleb. R.: Realisierung von Kommunikations- und Modellstrukturen bei grafischen Systemen.
Wiss. Z. Techn. Univ. Dresden (im Druck)

/4/ Tichatzky, R.: Bereitstellung der Voraussetzungen und Realisierung eines Dialog-im-Dialog-Systems am Digitalgeometrischen Arbeitsplatz.
Diplomarbeit, Sektion Mathematik, TU Dresden, Dresden 1979

eingegangen: 07. 11. 1979

Anschrift des Verfassers:

Dr. rer. nat. Rainer Ortleb
Technische Universität Dresden
Sektion Mathematik WB MKR
DDR-8027 Dresden
Mommsenstraße 13

Rostock. Math. Kolloq. 14, 65 - 81 (1980)

Thomas Pätz

Eine Methode zur automatischen Erzeugung von Overlaystrukturen

Zur Lösung von Problemen auf Rechnersystemen werden heute Systeme von Programmen oder Programmoduln verwendet. Programmoduln stehen dabei in bestimmten Relationen zueinander. Man spricht von Programmstrukturen, Programmsystemstrukturen, Datenabhängigkeiten, Steuerstrukturen, Programmaufrufstrukturen u.a.m. Gemeinsam ist allen Sprechweisen, daß wir eine Menge von Programmoduln haben und über dieser Menge orientierte Graphen, Strukturen betrachten.

Wir wollen von sogenannten Aufruf- oder Steuerstrukturen ausgehen. (Im weiteren wird nur von Aufrufgraphen gesprochen.) Die Knoten der Graphen sind Programmoduln, Unterprogramme oder Programmsegmente. Ein Bogen von A nach B bedeutet, daß A B aufruft.

Rechentechnisch besteht das Problem, einen gegebenen Programmgraph im Speicher zu realisieren, zu speichern. Dabei sind bestimmte Bedingungen zu erfüllen, und es entsteht die Aufgabe, den Graph nach bestimmten Gesichtspunkten zu zerlegen, um im Rahmen eines gegebenen Betriebssystems eine effektive Arbeit, z. B. durch Überlagerungstechniken, zu gewährleisten. In einigen Fällen, z. B. beim rechnergestützten Systementwurf, möchte man derartige Zerlegungen von Rechnersystemen automatisch durchführen lassen.

Wir betrachten einen Aufrufgraphen, der zerlegt werden soll, so daß er unter Beachtung der Bedingungen des Betriebssystems OS 4.1(MFT) im Hauptspeicher realisiert werden kann. Im nachfolgenden werden Aussagen über die Zerlegbarkeit bei der Nutzung einer statischen Überlagerungstechnik bewiesen und ein Zerlegungsalgorithmus angegeben.

Die dafür notwendigen graphentheoretischen Begriffe und Aussagen wurden /3 /entlehnt, dagegen sind die rechentechnischen Be-

griffe den üblichen Begriffsbildungen aus den Systemunterlagen des OS angepaßt (siehe z. B. /2/, /4/).

1. Rechentechnische Voraussetzungen

(1) Es existieren 4 Regionen.

(2) Es existiert eine eindeutige Wurzel des Aufrufgraphen, d.h. ein Programm, welches die Steuerung, Abarbeitungsbeginn und -ende enthält.

(3) Alle benötigten Systemprogramme werden der Wurzel zugeladen.

(4) Kein Programm tritt in der Overlaystruktur mehrfach auf.

(5) Ein Programm A kann die Bearbeitung an ein Programm B nur dann übergeben, wenn einer der nachfolgenden Fälle vorliegt:

(5) I. Wenn sich A und B in der gleichen Region befinden, dann gibt es in der Overlaystruktur eine Bahn von A nach B.

(5) II. Wenn A in einer Region i liegt und B in einer Region j, dann gilt $j > i$.

(5) III. Wenn B der Wurzel zugeladen ist, dann ist B durch den Aufruf von A in der Overlaystruktur aus jeder Region erreichbar.

(6) Der benötigte Kernspeicher S errechnet sich aus

$$S = \sum_{i=1}^{n} \max_{j} \left(\sum_{k_1} SP_{ij}^{k_1}, \ldots, \sum_{k_{n_j}} SP_{ij}^{k_n} j \right) + SSY_i;$$

SP_{ij}^{k} = Speicherplatzbedarf des k-ten Programms der j-ten Bahn der i-ten Region,

SSY_i = Speicherplatzbedarf der Systemroutinen und -informationen der i-ten Region

(siehe Bild 2, Tabelle 2 und Berechnung des Speicherplatzbedarfs).

(7) Parameter werden wie in Unterprogrammen vermittelt.

Die EDVA-gerechte Repräsentation einer Overlaystruktur erfolgt im OS 4.1.(MFT) in verallgemeinerter Hauptreihenfolge, wie das in /2/ und /4/ beschrieben ist (siehe dazu Bild 1 und 2 und Tabelle 1).

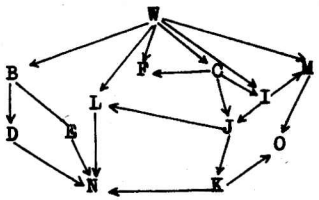


Bild 1: Ausgangsstruktur

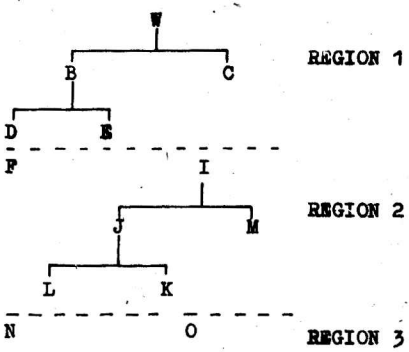


Bild 2: Overlaystruktur

```
OVERLAY 01
INSERT  B
OVERLAY 02
INSERT  D
OVERLAY 02
INSERT  E
OVERLAY 01
INSERT  C
OVERLAY 03 (REGION)
INSERT  F
OVERLAY 03
INSERT  I
OVERLAY 04
INSERT  J
OVERLAY 05
INSERT  L
OVERLAY 05
INSERT  K
OVERLAY 04
INSERT  M
OVERLAY 06 (REGION)
INSERT  N
OVERLAY 06
INSERT  O
```

Tabelle 1:
EDVA-gerechte Beschreibung der Overlaystruktur

Programmodul	Speicherplatzbedarfseinheiten	Programmodul	Speicherplatzbedarfseinheiten
W	10	J	17
B	12	K	12
C	16	L	30
D	3	M	120
E	17	N	50
F	107	O	30
I	4		

Tabelle 2: Speicherplatzbedarf der Programmoduln

Berechnung des Speicherplatzbedarfs:

$SP^1_{11} = SP^1_{12} = SP^1_{13} = SP(W) = 10$

$SP^2_{11} = SP^2_{12} = \text{----} = SP(B) = 12$

$SP^2_{13} = SP(C) = 16$

$SP^3_{11} = \text{----------} = SP(D) = 3$

$SP^3_{12} = \text{----} = SP(E) = 17$

S = max(10+12+3, 10+12+17, 10+16) + max(107, 51, 33, 124) + max (50, 30)

S = 213 Speicherplatz unter Berücksichtigung der Overlaystruktur ohne Systemprogramme und -informationen.

$S = \sum_K SP(K) = 428$ Speicherplatz ohne Overlaystruktur, Systemprogramme und -informationen.

2. Modell I

Definition 2.1: Die **Regionzahl** RE ist eine natürliche Zahl, die jedem Knoten B einer Aufrufstruktur wie folgt zugeordnet wird:
- Die Wurzel erhält die Regionzahl RE(B) = 1.
- Jeder Knoten B, dessen Vorgänger mit RE = r alle auf einer gemeinsamen Bahn nach B liegen und der keine Vorgänger mit RE > r hat, erhält RE(B) = r.
- Jeder Knoten B, dessen Vorgänger mit RE = r nicht alle auf einer gemeinsamen Bahn nach B liegen und der keine Vorgänger mit RE > r hat, erhält RE(B) = r + 1.

Definition 2.2: Die **maximale Regionzahl** MARE einer Teilstruktur ist eine natürliche Zahl, die der Teilstruktur zugeordnet wird, von der alle Knoten eine Regionzahl haben. Die maximale Regionzahl ist gleich der maximalen in der Struktur auftretenden Regionzahl:

$$MARE := \max_{h}(RE(h)),\ h \in \text{Menge aller Knoten.}$$

Definition 2.3: Die **minimale Regionzahl** MIRE einer Teilstruktur ist eine natürliche Zahl, die der Teilstruktur zugeordnet wird, von der alle Knoten eine Regionzahl haben. Die minimale, in der Struktur auftretende Regionzahl ist gleich der minimalen Regionzahl:

$$MIRE := \min_{h}(RE(h)),\ h \in \text{Menge aller Knoten.}$$

Definition 2.4: Die **Regionalzahl** REL einer Teilstruktur ist eine natürliche Zahl, die der Teilstruktur zugeordnet wird, falls MIRE = MARE ist. Dann ist REL = MIRE = MARE.

Wir benötigen den Begriff der maximalen Bahn nach B. Das ist eine Bahn t nach B, deren Länge L(t) maximal ist. Die Länge einer Bahn ist gleich der Anzahl der Knoten der Bahn.

Lemma 2.1: Eine konturfreie (Kontur entspricht der geschlossenen Bahn in /3/) Teilstruktur mit REL $\neq$ 0 ist eine Struktur, in der zu jedem Knoten außer den Wurzeln genau eine maximale Bahn existiert, die alle Vorgänger dieses Knotens enthält.

Beweis: Annahme: Es existieren 2 verschiedene maximale Bahnen b und c zu einem Knoten A. Die Bahn b bestehe aus $B_t,B_{t-1},\ldots,B_1,A$, die Bahn c aus $C_t,\ldots,C_1,A$.

Die Knoten C_1 und B_1 sind gleich, da es sonst eine Bahn zwischen C_1 und B_1 bzw. B_1 und C_1 geben müßte (nach Definition 2.1), d. h., es existiert eine Bahn d aus $C_t,\ldots,C_1,\ldots,B_1,A$ bzw. $B_t,\ldots,C_1,A$ mit $L(d) > L(b) = L(c)$. Das widerspricht der Annahme, daß b und c maximale Bahnen sind.
Folglich sind die Bahnen b' mit $B_t,\ldots,B_2,B_1$ und c' mit $C_t,\ldots,C_2B_1$ auch maximale Bahnen zu B_1.
Analog sind auch die Knoten C_i und B_i $(t \geq i \geq 2)$ gleich.
Daraus folgt, daß genau eine maximale Bahn zu A existiert.
Annahme: Es existiert ein Knoten V, der Vorgänger von A ist und nicht in der maximalen Bahn b liegt. Dann gibt es eine Bahn e mit $V,E_1,\ldots,E_L,B_1$ (nach Def. 2.1). (Es gibt keine Bahn f von B_1 nach V, da es andernfalls eine Bahn f' mit $B_t,\ldots,B_1,f,A$ und $L(f') > L(b)$ gibt, was aber der Voraussetzung, daß b maximale Bahn ist, widerspricht.) Diese Bahn e ist ab einem Knoten E_j (einschließlich B_1) in der Bahn b enthalten, d. h., alle Knoten der Bahn e ab E_j sind in der Bahn b vorhanden.
O.B.d.A. sei j = 1, d. h. $E_1 = B_i$, und es existiert kein $B_k = V$.
Damit sind B_{i-1} und V Vorgänger von B_i. Analog fortgesetzt, erkennt man, daß es keinen solchen Knoten V geben kann, der Vorgänger von A ist und nicht in b enthalten ist.

Satz 2.1: Wenn in einer einfachen Hecke mit genau einer Wurzel alle Kanten zwischen Knoten unterschiedlicher Regionzahl und danach alle nicht in einer maximalen Bahn liegenden Kanten gelöscht werden, dann zerfällt diese in genau einen Baum mit RRL = 1 sowie je einen Wald zu jeder weiteren auftretenden Regionalzahl.

Bemerkung: Im Bild 3 liegen z. B. die Kanten zwischen W und L, W und I sowie C und J in keiner maximalen Bahn.

Beweis: Durch Löschen der Verbindungen zwischen Knoten unterschiedlicher Regionzahl entstehen Strukturen mit Regionalzahl. Nach Lemma 2.1 ist jeder Knoten (außer den Wurzeln) einer Struktur mit Regionalzahl Zielpunkt genau einer maximalen Bahn, d. h., jeder Knoten hat nur Vorgänger aus dieser maximalen Bahn, die wiederum durch die maximale Bahn mittelbare Vorgänger sind. Durch das Löschen der nicht in der maximalen Bahn liegenden Kanten hat jeder Knoten höchstens einen Vorgänger, d. h., da die Struktur auch konturfrei ist, diese Struktur ist ein Wald.
Annahme: Es existieren zwei Bäume mit RÉL = 1, d. h., es existieren zwei Wurzeln mit RÉ = 1. Das ist aber ein Widerspruch zur Voraussetzung des Satzes.

Satz 2.2: Jede einfache Hecke mit genau einer Wurzel und MARE ≤ 4 kann als Overlaystruktur durch die Overlaytechnik des OS 4.1. beschrieben werden.

Beweis: Durch das Löschen der Kanten nach Satz 2.1 entstehen Wälder, die in Hauptreihenfolge aufschreibbar sind. Nach der Voraussetzung des Satzes ist die rechentechnische Voraussetzung (2) erfüllt. Indem jede Struktur mit RÉL = i in der Region i dargestellt wird, ist auch die rechentechnische Voraussetzung (5) erfüllt, denn Knoten von gelösten Aufrufkanten liegen entweder in verschiedenen Regionen (Fall (5)II) oder sind in einer gemeinsamen maximalen Bahn, d. h., es existiert eine Bahn zwischen den Knoten (Fall (5)I).

Satz 2.3: Strukturen mit den Eigenschaften aus Satz 2.2 und MARE > 4 lassen sich darstellen, indem mehrere Knoten zu einem Segment verschmolzen werden.

Beweis: Alle Knoten mit RÉ > 4 werden der Wurzel zugeladen und sind dann nach Voraussetzung (5)I aufrufbar.

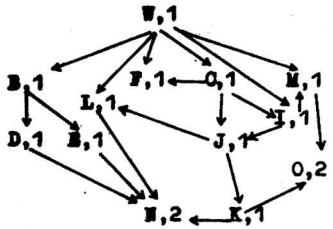


Bild 3:
Ausgangsstruktur aus
Bild 1 mit Regionzahlen
nach Modell I

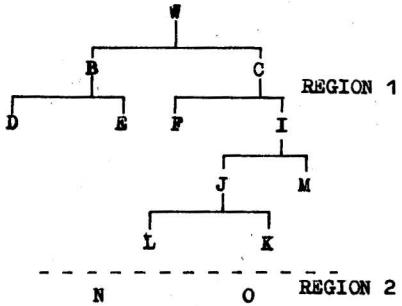


Bild 4: Overlaystruktur

Bemerkungen:

1. Bei Aufrufbäumen bleibt die Baumstruktur durch die Anwendung der Sätze unverändert erhalten und wird so als Overlaystruktur übernommen.

2. Da die Segmentanzahl meist begrenzt ist, z. B. im OS 4.1. (MFT) 255, können ohne Speicherplatzverlust auch Knoten, die nur einen Nachfolger haben, mit diesen verschmolzen werden.

3. Bei Konturen innerhalb der Aufrufstruktur, falls nicht prinzipiell ausgeschlossen, können diese in einem Segment verschmolzen werden. Danach ist dieses Modell wieder anwendbar.

4. Ein Algorithmus zur Erzeugung einer Overlaystruktur nach Modell I zerfällt in folgende 4 Bestandteile:

a) Regionzahlzuweisung,
b) Löschen der Kanten zwischen Knoten unterschiedlicher Regionzahl,
c) Löschen der Kanten, die in keiner maximalen Bahn liegen,
d) Beschreibung der Overlaystruktur.

3. Modell II

Das Modell I entspricht den im OS 4.1.(MFT) festgelegten Prinzipien bis auf exklusive Aufrufe. (Ein exklusiver Aufruf tritt z. B. auf, wenn in der Overlaystruktur aus Bild 4 D von J aufgerufen wird.) Da diese in der Praxis aber häufig zu Fehlern führen, kann auf solche Aufrufe im Modell verzichtet werden. Da das Modell I außerdem recht anspruchsvolle Teilalgorithmen benötigt, genügt in vielen Fällen ein vereinfachtes Modell. Ein solches ist das Modell II. Es entsteht aus Modell I, wenn in der Definition 2.1 nur maximal ein Vorgänger mit RE = r zugelassen wird.
Dadurch wird die rechentechnische Voraussetzung (5)I dahingehend eingeschränkt, daß der Aufruf von B durch A nur dann möglich ist, wenn in der Overlaystruktur ein Bogen von A nach B vorhanden ist, ansonsten ist B nur über eine Aufruffolge von A aus erreichbar. Z. B. ist in der Overlaystruktur aus Bild 4 der Aufruf von E durch W nach Modell I möglich, aber im Modell II benötigt man dazu die Aufruffolge W, B, E.

3.1. Beschreibung des Modells II

Durch den Übergang von Modell I zu Model II lassen sich einige Definitionen und Sätze einfacher formulieren.

Definition 3.1: Die Regionzahl RE eines Knotens ist eine natürliche Zahl, die jedem Knoten einer Aufrufstruktur wie folgt zugeordnet wird:
- Die Wurzel erhält die Regionzahl RE = 1.
- Jeder Knoten, der genau einen Vorgänger mit RE = r und keinen mit RE > r hat, erhält die Regionzahl RE = r.
- Jeder Knoten, der mehr als einen Vorgänger mit RE = r und keinen mit RE > r besitzt, erhält die Regionzahl RE = r + 1.

Die Definitionen von MARE, MIRE und REL für das Modell II sind äquivalent mit denen aus Abschnitt 2.

Lemma 3.1: Eine konturfreie Struktur mit Regionalzahl REL $\neq$ 0 ist ein Wald.

Beweis: Eine konturfreie Struktur habe eine Regionalzahl, d. h. MIRE = MARE. Das bedeutet, alle Knoten haben die gleiche Regionzahl. Daraus folgt, jeder Knoten hat höchstens einen Vorgänger. Wegen der Konturfreiheit erhalten wir einen Wald.

Satz 3.1: Wenn alle Verbindungen zwischen Knoten unterschiedlicher Regionzahl einer einfachen Hecke mit genau einer Wurzel gelöst werden, so zerfällt diese in genau einen Baum mit REL =1 sowie je einen Wald zu jeder weiteren auftretenden Regionalzahl.

Beweis: Durch Löschen der Kanten zwischen Knoten unterschiedlicher Regionzahl entstehen Strukturen mit Regionalzahl. Nach Lemma 3.1 sind diese Strukturen Wälder.
Annahme: Es existieren zwei Bäume mit REL = 1. Daraus folgt, es existieren zwei Wurzeln mit RE = 1. Das ist aber ein Widerspruch zur Voraussetzung.

Satz 3.2: Jede einfache Hecke mit genau einer Wurzel und MARE ≤ 4 kann als Overlaystruktur durch die Overlaytechnik des OS 4.1.(MFT) beschrieben werden.

Beweis: Durch Löschen der Verbindungen nach Satz 3.1 entstehen Wälder, deren Knoten in Hauptreihenfolge aufschreibbar sind. Die rechentechnische Voraussetzung (2) ist nach Voraussetzung erfüllt. Indem jede Struktur mit REL = i in der Region i dargestellt wird, ist auch die rechentechnische Voraussetzung (5) erfüllt, denn Knoten von gelösten Aufrufkanten liegen immer in verschiedenen Regionen.

Der Satz 2.3 und die Bemerkungen 1. bis 3. gelten auch hier im vollen Wortlaut.

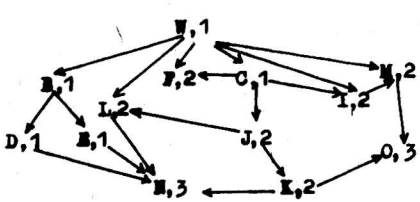


Bild 5:
Ausgangsstruktur aus Bild 1 mit Regionzahl nach Modell II
(Overlaystruktur s. Bild 2)

3.2. Algorithmus zur Erzeugung von Überlagerungsstrukturen

Der nachfolgende Algorithmus zur automatischen Aufstellung von Überlagerungsstrukturen basiert auf den Sätzen, Definitionen und den Bemerkungen 2. und 4. aus dem Abschnitt 3.1.
Für den Algorithmus wird eine Tabelle mit den Spalten PGN - Programmname, RE - Regionzahl (am Anfang = 0), M - Anzahl der Nachfolger, NNR bzw. HG - Hilfsgrößen (am Anfang = 10 bzw. = 0) für den Algorithmus und C_i - Nachfolgernamen benötigt.
Der erste Programmname in der Tabelle ist der Name der Wurzel, der letzte NIL.

PGN	RE	NNR	HG	M	C_1	C_2	C_3	C_4	C_5
W	0	10	0	4	A	D	B	F	
I	0	10	0	0					
A	0	10	0	2	C	D			
B	0	10	0	2	E	F			
F	0	10	0	1	H				
H	0	10	0	1	I				
E	0	10	0	2	G	H			
D	0	10	0	2	G	H			
C	0	10	0	1	G				
G	0	10	0	1	I				
NIL									

Tabelle 3: Eingabetabelle

Das mit Tabelle 3 angegebene Beispiel entspricht der Aufrufstruktur aus Bild 6. Am Ende des Abschnitts ist das Ergebnis der Anwendung des Algorithmus auf dieses Beispiel angegeben.

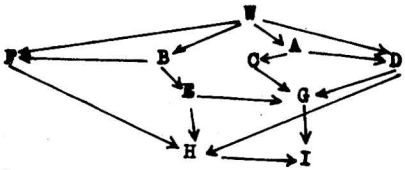


Bild 6: Aufrufstruktur

Hilfsgrößen sind NNRH, MN, REZ, MNI : INTEGER,
K1, K2, KW : Keller für Programmnamen,
GM, GM2, ORN, HILF, MERK : Größen, in denen jeweils ein Programmname abgespeichert werden kann.

Neben den Wertzuweisungs- und Arithmetikoperationen treten die Kelleroperationen auf, wie Einkellern und Auskellern, die durch ⇐ bzw. ⇒ symbolisiert werden, und Tabellenoperationen VOR(GM) und NEXT(GM). Den Vorgänger in der Tabelle (PGN-Spalte) des Programmnamens, der in GM gespeichert ist, ermittelt VOR(GM) und trägt ihn in GM ein. NEXT(GM) bestimmt den Nachfolger, z. B. GM := D; NEXT(GM) ergibt, daß C in GM steht.
Der sich anschließende Algorithmus zur Erzeugung einer Overlaystruktur zerfällt in Regionzahlzuweisung, Löschen der Kanten und Beschreibung der Overlaystruktur.

```
(* ZUORDNUNG DER REGIONZAHL *)
(* Anfangswerte herstellen *)
GM := PGN(1); RE( GM ) := 1; K1 ⇐ NIL; MN := 1;
(* nächster Knoten *)
```

```
3 : NNR( GM ) := NNR( GM )+1; MN := NNR( GM )-10;
    IF MN ≤ M( GM )
       THEN BEGIN GM2 := C_MN( GM ); GOTO 2 END
       ELSE BEGIN GM2 ⇐ K1;
            IF GM = NIL
               THEN BEGIN K1 ⇐= GM2; GOTO 5 END
               ELSE GOTO 3;
            END;
    (* Regionzahlzuweisung *)
2 : IF RE( GM ) < RE( GM2 ) THEN GOTO 3;
    IF RE( GM ) > RE( GM2 )
       THEN BEGIN RE( GM2 ) := RE( GM );
                  HG( GM2 ) := RE( GM )
            END;
       ELSE BEGIN
            IF HG( GM2 ) < RE( GM )
               THEN HG( GM2 ) := RE( GM )
               ELSE BEGIN HG( GM2 ) := RE( GM );
                          RE( GM2 ) := RE( GM2 )+1
                    END;
            END;
    NNR( GM2 ) := 10; K1 ⇐= GM; GM := GM2; GOTO 3;
    (* Ende der Zuweisung *)

    (* STREICHEN DER VERBINDUNGEN, BESTIMMEN UND SORTIEREN
       DER WURZELN *)
    (* Festlegung der Anfangswerte *)
5 : K2 ⇐ NIL; KW ⇐ NIL; GM := PGN(1);
    IF M( GM ) = 0 THEN GOTO 17 ELSE NNRH := 1;
    (* Lösen der Kanten und Nachfolgerverdichtung *)
12 : IF RE( C_NNRH( GM )) = RE( GM )
       THEN GOTO 13
       ELSE M( GM ) := M( GM )-1;
    IF NNRH > M( GM )
       THEN GOTO 14
       ELSE FOR MNI := NNRH TO M( GM )
                DO C_MNI( GM ) := C_MNI+1( GM );
    GOTO 12;
    (* Setzen der Hilfsgrößen und weitere Nachfolger *)
```

```
13 : NNR( C_NNRH( GM )) := 0; NNRH := NNRH+1;
     IF  NNRH > M( GM ) THEN GOTO 14 ELSE GOTO 12;
     (* weiteres Programm *)
14 : NEXT( GM );
     IF GM = NIL THEN GOTO 15;
     IF M( GM ) = 0 THEN GOTO 14;
     NNRH := 1;GOTO 12;
     (* Aussortieren der Wurzeln *)
15 : VOR( GM );
     IF NNR( GM ) ≠ 0 THEN BEGIN K1 ⇐ GM; NNR( GM ) := 0 END;
     IF GM ≠ PGN(1) THEN GOTO 15;
     (* Einsortieren der Wurzeln in den Wurzelkeller *)
     GM ⇐ K1; GM2 ⇐ K1;
     IF GM = NIL THEN GOTO 28;
110 : IF GM2 = NIL THEN BEGIN KW ⇐ GM; GOTO 111 END;
 17 : IF RE( GM ) ≥ RE( GM2 ) THEN BEGIN K2 ⇐ GM2;
        GM2 ⇐ K1 END ELSE BEGIN K2 ⇐ GM; GM ⇐ K1 END;
     IF GM = NIL
        THEN GOTO 19
        ELSE IF GM2 = NIL THEN GOTO 19 ELSE GOTO 17;
 19 : IF GM = NIL THEN KW ⇐ GM2 ELSE KW ⇐ GM;
     K1 ⇐ NIL; GM ⇐ K2; GM2 ⇐ K2;
     IF GM = NIL THEN GOTO 111;
     IF GM2 = NIL THEN BEGIN KW ⇐ GM; GOTO 111 END;
112 : IF RE( GM ) ≥ RE( GM2 )
        THEN BEGIN K1 ⇐ GM2; GM2 ⇐ K2 END
        ELSE BEGIN K1 ⇐ GM; K2 ⇒ GM END;
     IF GM2 = NIL
        THEN GOTO 11
        ELSE IF GM2 = NIL THEN GOTO 11 ELSE GOTO 112;
 11 : K2 ⇐ NIL;
     IF GM = NIL THEN KW ⇐ GM2 ELSE KW ⇐ GM;
     GM ⇐ K1;
     IF GM = NIL THEN GOTO 111 ELSE GM2 ⇐ K1; GOTO 110;
111 : K2 ⇐ NIL
     (* Ende des Streichens der Kanten und Wurzelbestimmung
        und -sortierung *)
```

```
     (* BESCHREIBUNG DER OVERLAYSTRUKTUR *)
     GM ⇐ KW; REZ := RE( GM );
     IF M( GM ) = 0 THEN GOTO 28;
     WHILE M( GM ) = 1 DO GM := C1( GM );
     (* Nachfolgerbestimmung *)
22 : WHILE M( GM ) = 1
          DO BEGIN WRITE('INSERT', C1( GM ));
                   GM := C1( GM )
             END;
     NNRH := NNR( GM )+1;
     IF NNRH > M( GM ) THEN GOTO 24;
     K1 ⇐ GM; ORN := GM; NNR( GM ) := NNRH;
     GM := C_NNRH( GM ); HILF := GM;
     (* Ausgabe *)
23 : WRITE('OVERLAY', ORN);
     WRITE('INSERT', GM);
     GOTO 22;
     (* Vaterbestimmung *)
24 : GM ⇐ K1;
     IF GM ≠ NIL THEN GOTO 22;
     (* weitere Wurzeln *)
     K1 ⇐ GM; GM ⇐ KW;
     IF GM = NIL THEN GOTO 28;
     IF RE( GM ) > 4 THEN GOTO 28;
     IF RE( GM ) = REZ THEN GOTO 27;
     (* Wurzeln mit neuer Regionzahl *)
     MERK := HILF;
     REZ := RE( GM );
     WRITE('OVERLAY', MERK, '(REGION)');
     WRITE('INSERT',GM);
     HILF := GM;
     GOTO 22;
     (* Wurzel mit gleicher Regionzahl *)
27 : ORN := MERK;
     GOTO 23;
     (* Ende der Overlaystrukturbeschreibung *)
28 : END
```

PGN	RE	NNR	HG	M	C_1	C_2	C_3	C_4
W	1	2	0	2	A	B	F	F
I	3	0	3	0				
A	1	1	1	1	C	D		
B	1	1	1	1	E	F		
F	2	0	1	0	H			
H	3	1	2	1	I			
E	1	0	1	0	G	H		
D	2	1	1	1	G	H		
C	1	0	1	0	G			
G	2	0	2	0	I			
NIL								

Tabelle 4: Beispieltabelle nach dem Abarbeiten des Algorithmus

```
OVERLAY W
INSERT A
INSERT C
OVERLAY W
INSERT B
INSERT E
OVERLAY A(REGION)
INSERT F
OVERLAY A
INSERT D
INSERT G
OVERLAY F(REGION)
INSERT H
INSERT I
```

Tabelle 5: Ergebnis

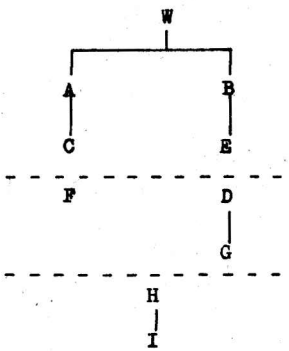


Bild 7: Overlaystruktur

Literatur

/1/ Gendt, G.: Aufbau und Zerlegung von Programmsystemen in Verbindung mit dem DIGRA 73-System. Diss. (A), Wilhelm-Pieck-Universität, Rostock 1974

/2/ Haupt, D., u. a.: Betriebssystem OS/ES Jobsteuerung/Ein- und Ausgabe. Leipzig 1977

/3/ Sachs, H.: Einführung in die Theorie der endlichen Graphen I. Leipzig 1970

/4/ Systemunterlagen - Dokumentation, Programmverbinder und Lader, Robotron C 5513-0001-1, Dresden 1975

eingegangen: 07. 11. 1979

Anschrift des Verfassers:

Dipl.-Math. Thomas Pätz
Wilhelm-Pieck-Universität Rostock
Rechenzentrum
DDR-2500 Rostock
Albert-Einstein-Straße 21

Rostock. Math. Kolloq. 14, 83 - 97 (1980)

Adolf Kotzauer
Bodo Urban

Unterprogrammtechnik auf strukturierten Informationen

Zusammenfassung

Bei der Behandlung von Problemen, die für ihre Lösung Datenstrukturen verwenden, treten bei der Nutzung von Unterprogrammen Besonderheiten in der Parametervermittlung auf. Die Ursachen liegen dabei in den vielfältigen strukturellen Beziehungen, die zwischen den Parametern bestehen können. Es werden die Besonderheiten aufgezeigt und die Vorgehensweise ihrer Behandlung durch Formulierung von Algorithmen angegeben.

1. Einführung

Für viele Anwendungen der EDV (z. B. Formelmanipulation, graphische Datenverarbeitung) ist es notwendig, mit Datenstrukturen zu arbeiten. Zur Lösung von Problemen aus diesen Gebieten müssen spezielle problemspezifische Datenstrukturen aufgebaut werden. Dazu sind Hilfsmittel notwendig, die oft Bestandteile der zugehörigen speziellen Fachsprache sind. Die Fachsprache ist in der Regel eine problemspezifische Erweiterung einer als Hostsprache fungierenden höheren Programmiersprache (z. B. DIGRA als FORTRAN-Erweiterung für die graphische Datenverarbeitung /4/, LISP-Dialekte für die Formelmanipulation /5/).

Die Behandlung umfangreicher Probleme erfordert eine Zerlegung in Teilprobleme, die zunächst unabhängig voneinander bearbeitet und gelöst werden müssen. Folglich muß eine Fachsprache die Möglichkeit besitzen, Teilprogramme bzw. Unterprogramme zu formulieren. Da die einzelnen Unterprogramme ebenfalls auf Datenstrukturen arbeiten, sind bei Aufrufen solcher Unterprogramme Besonderheiten zu beachten. Diese Besonderheiten haben ihre Ursachen in den vielfältigen strukturellen Beziehungen, die zwischen den Parametern bestehen können.

Die Hilfsmittel für diese Problematik, die moderne höhere Programmiersprachen wie PL/1 und ALGOL 68 anbieten, sind einerseits für die Definition der benötigten Datenstrukturen nicht ausreichend (besonders bei Strukturänderungen im Dialog), und andererseits beinhalten sie Restriktionen beim Aufruf von Teil- bzw. Unterprogrammen in der Parametervermittlung, die für einen Anwender nicht einsehbar und deshalb nicht akzeptierbar sind.

Ferner gilt zu beachten, daß als Hostsprache noch oft Programmiersprachen wie z. B. FORTRAN verwendet werden, die keinerlei Hilfsmittel zur Definition und Arbeit mit Datenstrukturen aufweisen. Deshalb ist die Kenntnis und die Behandlung der Problematik der strukturierten Parameter in der Unterprogrammtechnik, d. h. von Parametern, deren Werte Strukturen sind, für den Programmierer entsprechender Software notwendig, denn vom Anwender kann keine andere Behandlung der strukturierten Parameter verlangt werden, als die Form der Parameterbehandlung der Hostsprache.

In der Arbeit werden folglich wesentliche Punkte der Unterprogrammtechnik auf Strukturen diskutiert. Dabei wird in den Untersuchungen von der Datenstruktur ausgegangen und nicht von einer höheren Programmiersprache als Hostsprache.

Für die Behandlung der strukturierten Parameter beim Unterprogrammaufruf und -rücksprung werden in der Arbeit Algorithmen angegeben.

1.1. Beispiel

Die Besonderheiten der Parametervermittlung sollen zunächst am Beispiel der Positionierung von Maschinen in einer Werkhalle aufgezeigt werden. Dazu wird folgendes angenommen:

1. Im Hauptprogramm (Name: MAUFST) werden die benötigten Maschinen als Graphen definiert.
2. Im Unterprogramm (Name: MPOS) wird die Maschinenhalle als Graph definiert und die Positionierung der Maschinen in der Halle ausgeführt.

Es wird mit den vier Maschinen AAGR (Antriebsaggregat), DRBNK (Drehbank), OELPMP (Ölpumpe) und BMASCH (Bohrmaschine) gearbeitet, die in der Werkhalle MHAL positioniert werden sollen. Dabei soll OELPMP an DRBNK gebunden sein, d. h., im Graph von DRBNK ist ein Verweis auf OELPMP vorhanden. Ferner soll AAGR aus MOT (Motor) und STAND (Gestell) zusammengesetzt sein. Das Unterprogramm MPOS hat folgende formale Parameter:

GR1, GR2, GR3, GR4 — graphische Parameter; charakterisieren die vier zu positionierenden Maschinen.

MH — graphischer Parameter; liefert die Maschinenhalle mit den positionierten Maschinen als Graph.

XFELD — numerischer Parameter (Feldname); enthält Angaben über die Abmessungen der Maschinenhalle.

Zur Angabe des Programms wird die graphische Fachsprache DIGRA (s. /4/) verwendet.

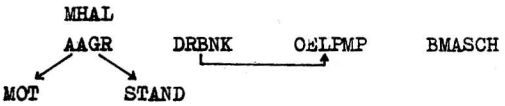
```
      DIGRA PROGRAM MAUFST
            :
      Definition der Graphen von AAGR, DRBNK, OELPMP, BMASCH,
      MOT und STAND
            :
      DIGRA CALL MPOS (AAGR, DRBNK, OELPMP, BMASCH, MHAL,
      NUMFELD)
            :
      Ausgabe von MHAL auf Bildschirm
            :
      END  MAUFST

      DIGRA SUBROUTINE MPOS (GR1, GR2, GR3, GR4, MH, XFELD)
C     DEFINITION DER MASCHINENHALLE
            :
      PF1 = ...  } Aufbau von zwei Stützpfeilern PF1, PF2
      PF2 = ...  } als Teilgraphen von MH
            :                          [Positionierung von PF1,
      MH = ... F1*PF1.F2*PF2 ...  ---  [PF2 in der Halle (F1, F2
            :                          [sind aus Daten von
                                       [XFELD zusammengestellt)
```

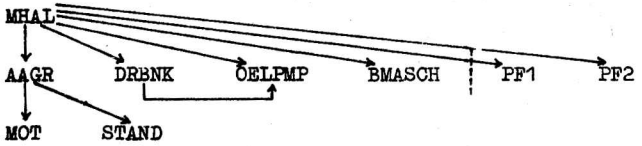
```
C       POSITIONIERUNG DER MASCHINEN
                  :
                  :  }   Berechnung der Positionen POS1, POS2,
                  :  }   POS3, POS4 aus Daten von XFELD
        MH = MH.POS1*GR1.POS2*GR2.POS3*GR3.POS4*GR4
                  :
                  :
        RETURN
        END  MPOS
```

Zwischen den Graphen in MAUFST bestehen folgende strukturelle Beziehungen (Pfeile geben Verweise an):

Unmittelbar vor dem Aufruf von MPOS



Unmittelbar nach der Rückkehr aus MPOS



1.2. Besonderheiten

Die aus der Unterprogrammtechnik bekannte Zuordnung von aktuellen zu formalen Parametern durch Angabe in der jeweiligen Parameterliste bleibt auch hier erhalten. Jedoch treten bei der Vermittlung der Werte ins und vom Unterprogramm folgende Besonderheiten auf:

- Die Werte der aktuellen Parameter können Verweise auf Größen enthalten, die nicht im Unterprogramm definiert sind.
  Im obigen Beispiel gilt dies bei dem aktuellen Parameter AAGR für die Größen MOT und STAND.
- Analoges kann nach dem Rücksprung aus einem Unterprogramm bei aktuellen Parametern auftreten, denen im Unterprogramm ein Wert zugewiesen wurde.

Im obigen Beispiel gilt dies für MHAL mit den Größen PF1 und PF2.

- Beim Unterprogrammaufruf sind evtl. vorhandene strukturelle Beziehungen zwischen aktuellen Parametern auf die entsprechenden formalen Parameter zu übertragen.
  Im obigen Beispiel ist die Beziehung zwischen DRBNK und OELPMP auf GR2 und GR3 zu übertragen.
- Analoges gilt für den Rücksprung bei vorhandenen Beziehungen zwischen formalen Parametern.
  Im obigen Beispiel trifft dies u. a. auf MH und GR1 zu.
- Beim Verändern von Größen, deren Wert in einem Unterprogramm ermittelt wurde, kann in der Struktur Ballast (Müll) entstehen.
  Im obigen Beispiel sind die Werte von PF1 und PF2 Müll, wenn in MAUFST nach Abarbeitung von MPOS der Wert von MHAL gelöscht wurde.

2. Parametervermittlung

Für die weiteren Untersuchungen wird nur die Parametervermittlung für strukturierte Werte betrachtet. Dabei wird gefordert, daß die Arbeit mit Strukturen im Unterprogramm in gleicher Weise abläuft wie im Hauptprogramm, d. h., für die Strukturarbeit sind jeweils die gleichen Grundfunktionen verwendbar.

2.1. Begriffe

Begriffe aus der Theorie der Datenstrukturen:

Item	: Kleinste Einheit in der Datenstruktur; kann Daten enthalten oder Verweise auf Strukturelemente.
Strukturelement	: Lineare Liste von Items; besitzt einen Identifikator.
Identifikator	: Bezeichnung eines Strukturelementes.
Identifikatoritem	: Speichermäßge Realisierung eines Identifikators.

Zugriffsstruktur : Struktur der Identifikatoritems zur Gewährleistung eines effektiven Zugriffs auf einen Identifikator und damit auf das entsprechende Strukturelement.

Zugriff über den Namen : Zugriff auf ein Strukturelement über die Zugriffsstruktur, d. h. über den Identifikator.

Zugriff über die Adresse: Direkter Zugriff auf das Identifikatoritem, d. h. ohne Nutzung der Zugriffsstruktur.

Abkürzungen und Begriffe zur Beschreibung der Unterprogrammtechnik:

$NR(TP_i)$ - Namensraum des Teilprogramms TP_i; gibt die Menge aller im TP_i definierten Identifikatoren an.

$SR(TP_i^j,\tau)$ - Strukturraum des Teilprogramms TP_i bei der j-ten Abarbeitung zum Zeitpunkt τ; gibt die Menge aller Identifikatoren an, auf die in TP_i zum Zeitpunkt τ über den Namen oder über die Adresse zugegriffen werden kann.
Dabei ist

$$\tau \in [\tau_A, \tau_R]$$

mit

τ_A als Zeitpunkt des j-ten Aufrufs des Teilprogramms und

τ_R als Zeitpunkt des Rücksprungs aus dem Teilprogramm.

Für viele Untersuchungen ist die Annahme $\tau_A=0$ ausreichend.

$W(I,\tau)$ - Wertfunktion $W : NR \times T \rightarrow \mathcal{P}(SR) \times T$; gibt zum Zeitpunkt τ des arbeitenden Teilprogramms TP_i (j-ter Aufruf) für einen Identifikator $I \in NR(TP_i)$ die Menge der Identifikatoren aus $SR(TP_i^j,\tau)$ an, für die gilt:

$W(I,\tau) = \{I' \mid I' \in SR(TP_i^j,\tau) \wedge I \rightarrow I'\}$.
Dabei bedeutet $I \rightarrow I'$ (I verweist auf I'), daß ausgehend von I über dessen Strukturelement ein Zugriff über die Adresse (evtl. auch über mehrere Stufen) auf I' möglich ist.

$R(ID,\tau)$ - Referenzfunktion $R : \mathcal{P}(NR) \times T \rightarrow \mathcal{P}(SR) \times T$; gibt zum Zeitpunkt τ des arbeitenden Teilprogramms TP_i (j-ter Aufruf) für die Menge ID mit $ID \subseteq NR(TP_i)$ die Menge der Identifikatoren aus $SR(TP_i^j,\tau)$ an, für die gilt:
$R(ID,\tau) = \{I \mid I \in SR(TP_i^j,\tau) \wedge I \in W(I'.\tau)$ für ein $I' \in ID\}$.

$P(AP)$ - Parameterfunktion $P : AID \rightarrow FID$.
Dabei ist AID die Menge der Identifikatoren der aktuellen Parameter für den n-ten Aufruf von TP_m aus TP_i^j und
FID die Menge der Identifikatoren der formalen Parameter von TP_m.
P ist eine eineindeutige Funktion von AID auf FID, also:
$P(AP) = FP$ und $P^{-1}(FP) = AP$ mit $AP \in AID$ und $FP \in FID$.

Es gelten folgende Beziehungen:

$NR(TP_i) \subseteq SR(TP_i^j,\tau)$ für jedes j und τ;
$R(ID,\tau) \subseteq SR(TP_i^j,\tau)$ für beliebige $ID \subseteq NR(TP_i)$ und für jedes τ.

Die Anwendung auf das Beispiel in 1.1. ergibt:

$NR(MAUFST) = \{AAGR,MOT,STAND,DRBNK,OELPMP,BMASCH,MHAL\}$
$SR(MAUFST^1,\tau_A) = NR(MAUFST)$
$SR(MAUFST^1,\tau_R) = NR(MAUFST) \cup \{PF1,PF2\}$
$NR(MPOS) = \{GR1,GR2,GR3,GR4,MH,PF1,PF2\}$
$SR(MPOS^1,\tau) = NR(MPOS) \cup \{MOT,STAND\}$

Für die Betrachtungen des Aufrufs eines beliebigen Teilprogramms TP_m (n-ter Aufruf) von einem beliebigen Teilprogramm TP_i (j-ter Aufruf) und des Rücksprungs von TP_m^n nach TP_i^j sind folgende Mengen und Beziehungen interessant:

- Ist AID die Menge der Identifikatoren der aktuellen Parameter für den Aufruf von TP_m^n aus TP_i^j und FID die Menge der Identifikatoren der formalen Parameter von TP_m, dann gilt

  $AID \subseteq NR(TP_i)$,

  $FID \subseteq NR(TP_m)$,

  und es ist

  AVS = R(AID,τ^*), wobei τ^* der Zeitpunkt des Aufrufs von TP_m^n aus TP_i^j ist,

  AVS1= AVS $\cap$ AID,

  AVS2= AVS $\setminus$ AVS1,

  FVS = R(FID,τ_R), wobei τ_R der Zeitpunkt des Rücksprungs von TP_m^n nach TP_i^j ist.

  Wir bezeichnen AVS als die Menge der Identifikatoren der aktuellen Verweisstruktur und FVS als die Menge der Identifikatoren der formalen Verweisstruktur.

- Ist LID die Menge der Identifikatoren der lokalen Größen von TP_m, so gilt

  $LID \subset NR(TP_m)$,

  und es ist

  LID1 = LID $\cap$ FVS,

  LID2 = LID $\setminus$ LID1.

Für das Beispiel in 1.1. mit MAUFST = TP_i^j und MPOS = TP_m^n gilt:

AID = {AAGR,DRBNK,OELPMP,BMASCH,MHAL}
FID = {GR1,GR2,GR3,GR4,MH}
AVS = {MOT,STAND,OELPMP}
AVS1 = {OELPMP}
AVS2 = {MOT,STAND}

FVS = {GR1,GR2,GR3,GR4,PF1,PF2,MOT,STAND}
LID = {PF1,PF2}
LID1 = {PF1,PF2}
LID2 = Ø

2.2. Aufruf

Zur Vermeidung von Namenskonflikten bei der Abarbeitung von Unterprogrammen sollte die Zugriffsstruktur stets nur die Identifikatoren des Namensraumes des gerade aktiven Teilprogramms enthalten. Deshalb sind beim Aufruf von TP_m^n durch TP_i^j folgende Aktionen auszuführen:

A1: Konservierung des aktuellen Standes der Zugriffsstruktur von TP_i^j.

A2: Aufbau der Zugriffsstruktur von TP_m^n.

A3: Parametervermittlung von TP_i^j nach TP_m^n.

Für die Realisierung von A1 und A2 reichen im wesentlichen die Grundfunktionen der verwendeten Datenstruktur aus. Für die Ausführung von A3 sind folgende Schritte notwendig:

A3.1: Realisierung von P(AP) = FP.

A3.2: Vermittlung der Werte der Elemente von AID an die entsprechenden Elemente von FID.

A3.3: Übernahme der Elemente von AVS2 in $SR(TP_m^n, \tau_A)$, wobei τ_A der Zeitpunkt des n-ten Aufrufs von TP_m durch TP_i^j ist.

A3.4: Falls AVS1 $\neq$ Ø, ist die Strukturbeziehung, die dann zwischen Elementen von AID besteht, auf die entsprechenden Elemente von FID zu übertragen.

Dabei erfordert die Realisierung von A3.3 und A3.4 einen großen Aufwand, da die Ermittlung von AVS kostspielig ist. Bei der Vermittlung zwischen AID und FID (Schritt A3.2) sind zwei Varianten zu unterscheiden.

Variante 1: Vermittlung durch den Wert. Hier wird die vollständige Unabhängigkeit (keine gegenseitigen Verweisbeziehungen) von $SR(TP_i^j, \tau)$ und $SR(TP_m^n, \tau)$ verlangt. Dazu müssen neben den

Werten der Elemente aus AID in A3.3 auch die Werte der Elemente aus AVS2 kopiert werden. Das erfordert die Bestimmung von AVS = R(AID,τ_A). In A3.4 ist dann stets eine Strukturaufbereitung vorzunehmen, bei der einmal die evtl. bestehende Strukturbeziehung zwischen AID auf FID und zum anderen die Strukturbeziehung zwischen AID und AVS2 auf FID und FVS übertragen wird. Dafür ist ein sehr großer Aufwand notwendig. Außerdem entsteht durch das Kopieren von Strukturelementen Redundanz in der Datenstruktur.

Variante 2: Vermittlung durch den Namen. Es werden Referenzketten vom formalen Parameter FP zum entsprechenden aktuellen Parameter AP (P(AP)=FP) aufgebaut, über die der Zugriff zum Wert erfolgt (s. Bild 1(a)). Auf Elemente von AVS2 kann nach der Vermittlung stets über die Adresse zugegriffen werden. Die strukturellen Beziehungen zwischen den Elementen von AID bleiben bei der Vermittlung auch für FID erhalten, da die unmittelbare Referenz auf ein Strukturelement in allen Teilprogrammen stets vom gleichen Identifikatoritem erfolgt.

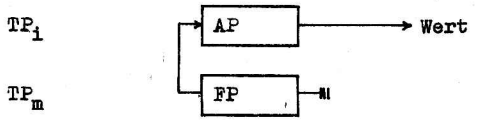
Bei tieferer Unterprogrammschachtelung können die Referenzketten recht lang werden, was dann eine starke Belastung für die Verarbeitungsprogramme bedeutet, da ein Zugriff über den Namen die Abarbeitung vieler Referenzen erfordert.

Um letzteres zu vermeiden, kann die Referenz zum Wert stets am Anfang der Referenzkette angebracht werden (s. Bild 1(b)). Dies hat jedoch für AVS1 $\neq \emptyset$ eine aufwendige Strukturaufbereitung zur Folge, da in geschachtelten Unterprogrammen Verweise auf einen Wert von unterschiedlichen Identifikatoritems der Referenzkette erfolgen. Dieser Nachteil kann mit folgender Strategie bei der Vermittlung beseitigt werden (s. auch Bild 1(c)):

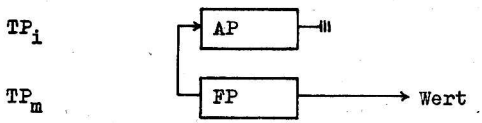
Es sei AP $\in$ AID und FP $\in$ FID mit P(AP) = FP, dann sind für jedes AP aus AID folgende Schritte notwendig:

1. Kopieren von AP in ein neues Identifikatoritem.
2. Aufbau von FP im "alten" Identifikatoritem von AP.
3. Anhängen des ("neuen") Identifikatoritems von AP an das von FP bzw. bei vorhandener Referenzkette Einhängen des ("neuen") Identifikatoritems von AP zwischen das von FP und das der restlichen Referenzkette.

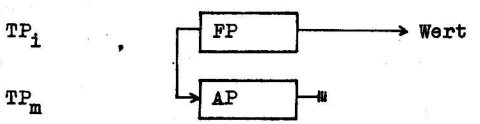
Damit befindet sich die Referenz zum Wert stets am Anfang der Referenzkette, und sie erfolgt für einen Wert in allen Teilprogrammen aus dem gleichen Identifikatoritem.



(a) Referenzkette mit Wert am Ende der Kette



(b) Referenzkette mit Wert am Anfang der Kette



(c) Aufbau von Referenzketten mit Umspeichern von Identifikatoren

Es gilt: $AP \in AID$ und $FP \in FID$ mit $P(AP) = FP$.

Bild 1: Vermittlung durch den Namen

2.3. Rücksprung

Beim Rücksprung vom TP_m^n ins TP_i^j sind folgende Aktionen auszuführen:

R1: Parametervermittlung von TP_m^n nach TP_i^j.

R2: Freigabe der Zugriffsstruktur von TP_m^n.

R3: Aktivieren der Zugriffsstruktur von TP_i^j.

Für die Realisierung von R2 und R3 sind im wesentlichen die Grundfunktionen der verwendeten Datenstruktur ausreichend. Für die Ausführung von R1 sind folgende Schritte notwendig:

R1.1: Vermittlung der Werte der Elemente von FID an die entsprechenden Elemente von AID.

R1.2: Übernahme der Elemente von FVS in $SR(TP_i^j, \tau_R)$, wobei τ_R der Zeitpunkt des Rücksprungs von TP_m^n nach TP_i^j ist.

R1.3: Falls $FVS \cap FID \neq \emptyset$, ist die Strukturbeziehung, die zwischen Elementen von FID besteht, auf die entsprechenden Elemente von AID zu übertragen.

Bei der in 2.2. vorgeschlagenen Variante der Vermittlung entfällt die Strukturaufbereitung (d. h. R1.3); auch die Werte der Elemente aus FVS, die in TP_i^j definiert sind, brauchen nicht berücksichtigt zu werden. Problematisch ist die Vermittlung der Werte der Elemente von LID1 nach $SR(TP_i^j, \tau_R)$. Möglich wäre eine Reduktion dieser Werte auf Strukturkonstanten, was jedoch sehr aufwendig wäre. Günstiger ist eine direkte Eingliederung dieser Werte ins TP_i^j. Um Müllentstehung bei Wertveränderungen zu vermeiden, sind spezielle Maßnahmen für die Elemente aus LID1 in der Zugriffsstruktur notwendig, so daß ein gezielter Zugriff auf sie möglich ist, d. h., beim Rücksprung wird LID von TP_i^j um LID1 aus TP_m^n erweitert.

Die Vermittlung von FID nach AID läuft nach folgendem Algorithmus ab:

1. Bestimmung von LID1, Herauslösen der Elemente von LID1 aus der Zugriffsstruktur von TP_m^n und Verbinden dieser Elemente zu einer linearen Liste.
2. Für jedes $FP \in FID$ mit P(AP) = FP gilt:
   2.1. Herauslösen von FP aus der Zugriffsstruktur,
   2.2. Kopieren von AP in das Identifikatoritem von FP,
   2.3. Freigabe des "alten" Identifikatoritems von AP.
3. Freigabe der Werte der Elemente von LID2.

4. Freigabe der Zugriffsstruktur von TP_m^n.

5. Aktivieren der Zugriffsstruktur von TP_i^j.

6. Anhängen der linearen Liste von LID1 an ein spezielles Element der Zugriffsstruktur von TP_i^j.

3. Realisierung

Die vorgestellten Ergebnisse basieren auf Erfahrungen bei der Nutzung der graphischen Unterprogrammtechnik des DIGRA 73 - Systems (s. /6/). Eine weitere Realisierung erfolgt auf einer algebraisch-graphischen Struktur. Diese Struktur ist ein Bestandteil des DIGRA 80 - Systems. Dieses interaktive graphische System befindet sich gegenwärtig an der Wilhelm-Pieck-Universität Rostock in Arbeit.

4. Schlußbemerkung

Obwohl die Problematik von uns auf Strukturen angewendet wurde, die für die Digitalgraphik genutzt werden, tritt sie nach unserer Meinung bei der Arbeit mit Strukturen generell auf. Sie wird immer dann benötigt, wenn für einen Anwendungskomplex voneinander unabhängige Programmbausteine gebraucht werden.
Die dargelegte Form der Unterprogrammtechnik auf Strukturen wird man nicht für jede Anwendungsaufgabe nutzen wollen, da relativ viel Zeit bei der Abarbeitung eines Unterprogramms für den Aufruf und den Rücksprung verwendet wird. Hier empfiehlt es sich, zwei Arten von Unterprogrammen, die internen und die externen, einzuführen. Bei internen Unterprogrammen sind der Namens- und Strukturraum des aufgerufenen und des aufrufenden Teilprogramms identisch, und es entfällt somit die Problematik der Parametervermittlung beim Aufruf und Rücksprung. Bei externen Unterprogrammen tritt die Parametervermittlung, wie sie in der Arbeit behandelt wurde, in Aktion.

Interne Unterprogramme kann man z. B. verwenden, wenn auf Grund bestimmter Steuerparameter (z. B. numerische Größen) im

Unterprogramm eine Struktur aufgebaut wird.
Werden noch Felder von strukturierten Größen zugelassen und können diese nebst ihren Elementen als Parameter auftreten, so sind im vorgeschlagenen Konzept einige Algorithmen zu erweitern.

Wir danken den Kollegen der Forschungsgruppe Digitalgraphik vom Rechenzentrum der Wilhelm-Pieck-Universität Rostock für die Diskussionen und Hinweise zur Problematik und Herrn Prof. Dr. H. Kiesewetter vom ZfR der Akademie der Wissenschaften für die Unterstützung und Hinweise bei der Behandlung der Thematik.

Literatur

/1/ Decker, W., und Gendt, G.: Routinen für die Realisierung der graphischen Unterprogrammtechnik. Bericht SM-DIGRA-72, Rostock 1974

/2/ Gendt, G.: Aufbau und Zerlegung von Programmsystemen in Verbindung mit dem DIGRA 73-System. Bericht SM-DIGRA-88, Rostock 1974

/3/ Kotzauer, A., und Urban, B.: Konzeption für die Unterprogrammtechnik im DIGRA 80-System. Bericht WPU-DIGRA-25, Rostock 1978

/4/ Kotzauer, A.: Die graphische Programmiersprache DIGRA. Wiss. Z. Univ. Rostock, Math.-Natur. Reihe 21, 8, 715 - 725 (1972)

/5/ Mätzel, K., Richter, R., und Schulz, M.: Symbolische algebraische Manipulation auf Rechenanlagen. Wiss. Z. Techn. Hochsch. Karl-Marx-Stadt 16, 3, 455 - 467 (1974)

/6/ Autorenkollektiv: Programmierhandbuch DIGRA 73. Schriftenreihe des Instituts für Schiffbau, Rostock 1974

eingegangen: 07. 11. 1979

Anschrift der Verfasser:

Dr. rer. nat. Adolf Kotzauer
Dipl.-Math. Bodo Urban
Wilhelm-Pieck-Universität Rostock
Rechenzentrum
DDR-2500 Rostock
Albert-Einstein-Straße 21

Rostock. Math. Kolloq. 14, 99 - 100 (1980)

Friedrich Wehmer

Zur Isomorphie gerichteter, konturfreier Graphen

Autorreferat der Dissertation

Es werden gerichtete, konturfreie Graphen (sogenannte Hecken) bezüglich Isomorphie untersucht. Die Einschränkung auf diese Klasse von Graphen ergibt sich u. a. aus Aufgabenstellungen in der Informationsverarbeitung.

In vier verschiedenen Verfahren wird das Isomorphieproblem von Hecken auf das Isomorphieproblem einer speziellen Klasse von paaren Graphen reduziert. Das erste Verfahren teilt alle Knoten einer Hecke auf der Grundlage ihrer Charakteristikvektoren (bestehend aus sechs Komponenten: Ein- und Ausgangsvalenz, Anzahl der Knoten und Bögen in den Bahnen, die in der Wurzel beginnen und im gegebenen Knoten enden bzw. im gegebenen Knoten beginnen und in einem der Gipfel der Hecke enden) niveauweise in Klassen ein. Die Knoten einer Klasse werden zu einem Makroknoten zusammengefaßt. Zwei Makroknoten sind adjazent, wenn zwischen ihren Knoten in der ursprünglichen Hecke Bögen existieren. Dadurch wird eine Makrohecke erzeugt. Falls beim schrittweisen Vergleich während der Erzeugung der Makrohecken M(H) und M(H') keine Aussage über die Isomorphie gemacht werden kann, erfolgt ein Isomorphievergleich der einander entsprechenden paaren Graphen der Makrohecken. Die Knotenmengen dieser paaren Graphen sind jeweils zwei adjazente Makroknoten (d. h. die Knoten, die zu diesen Makroknoten zusammengefaßt wurden), die Bogenmenge besteht aus den zwischen diesen Knoten existierenden Bögen. Wenn für zwei solcher paaren Graphen Nichtisomorphie festgestellt wird, folgt daraus die Nichtisomorphie von H und H', sonst sind H und H' isomorph.

In drei weiteren Verfahren erfolgt die Bildung von Makroknoten auf der Grundlage der Begriffe des maximalen Gerüstes (maximal im Sinne eines bestimmten Kodes für Wurzelbäume) bzw. der Auf-

fächerung von Hecken und längster Maximalbahnen (Bahnen mit maximaler Bogenanzahl zwischen der Wurzel und je einem Gipfel). Die Behandlung der Makrohecken erfolgt ähnlich wie oben beschrieben.

Im zweiten Teil der Arbeit werden zwei effektive Algorithmen zur Erzeugung aller paarweise nichtisomorphen $\mathfrak{T}$-Netze mit gegebener Bogenanzahl vorgestellt. Der erste Algorithmus basiert auf einem graphischen Modell der Darstellung von $\mathfrak{T}$-Netzen (Verwendung einer eineindeutigen Abbildung der $\mathfrak{T}$-Netze in die Klasse der Wurzelbäume), der zweite auf der $\mathfrak{T}$-Netz-Algebra (alle $\mathfrak{T}$-Netze werden mit Hilfe einer Bogenmenge U und den über U definierten Punkt- und Kommaoperationen dargestellt). Es werden effektive Algorithmen für die Feststellung der Isomorphie oder Nichtisomorphie zweier beliebiger $\mathfrak{T}$-Netze angegeben. Als Beispiel werden mit dem graphischen Modell alle paarweise nichtisomorphen ungerichteten $\mathfrak{T}$-Netze mit fünf Kanten erzeugt. Für den auf der $\mathfrak{T}$-Netz-Algebra aufbauenden Erzeugungsalgorithmus werden Programmablaufpläne und in einer speziellen Programmiersprache erstellte Programme entwickelt.

eingereicht: 07. 07. 1978 verteidigt: 23. 03. 1979

Gutachter: Prof. Dr. H. Kiesewetter (Rostock),
Prof. Dr. G. Burosch (Rostock),
Prof. Dr. O. B. Lupanov (Moskau)

Anschrift des Verfassers:

Dr. rer. nat. Friedrich Wehmer
Wilhelm-Pieck-Universität Rostock
Rechenzentrum
DDR-25 Rostock
Albert-Einstein-Straße 21