

Eine Implementierung von D A T A S

[Datenstrukturen in
Assoziativer Speicherung]

von

J. Encarnação - G. Weck

A 74-1

Institut für angewandte Mathematik
und Informatik der Universität des
Saarlandes

D-66 Saarbrücken
Bundesrepublik Deutschland

Januar 1974

Bericht Nr.: A 74-1

INHALTSVERZEICHNIS

0	Vorwort	
1.	<u>DATENSTRUKTUREN IN ASSOZIATIVER SPEICHERUNG</u>	1-1
1.1	<u>Datenstrukturen</u>	1-1
1.1.1	Allgemeines	1-1
1.1.2	Assoziative Datenstrukturen	1-2
1.2	<u>Adressierungsmöglichkeiten von Daten</u>	1-2
1.2.1	Explizite (direkte) Adressierung	1-2
1.2.2	Zeiger (Indirekte Adressierung)	1-3
1.2.3	Hash-Codierung	1-4
1.2.4	Implizite (assoziative) Adressierung	1-6
1.3	<u>Merkmale und Vergleich verschiedener assoziativer Datenstrukturen</u>	1-7
1.3.1	CØRAL	1-7
1.3.2	LEAP	1-8
1.3.3	SAM	1-8
1.3.4	DATAS	1-9
1.4	<u>Beschreibung des DATAS-Systems</u>	1-10
1.4.1	Die Komponenten des Systems	1-10
1.4.2	Entities	1-11
1.4.3	Triaden	1-12
1.4.4	Beispiel einer Assoziation in DATAS	1-13
1.4.5	Ausblick	1-16
2.	<u>DIE STRUKTUR DES SYSTEMS</u>	2-1
2.1	<u>Aufteilung in Gruppen von Unterprogrammen</u>	2-1
2.1.1	Forderungen an das System	2-1
2.1.2	Realisierungsweise	2-6
2.1.3	Bezeichnungen	2-7
2.1.4	Funktionelle Gruppen von Unterprogrammen	2-8
2.2	<u>Speicherstruktur</u>	2-11
2.2.1	Das Paging-System	2-11
2.2.1.1	Die Seitentabelle (Page-Tabelle)	2-12
2.2.1.2	Der Seitentransport (Page-Routine)	2-14
2.2.2	Die Strukturen der einzelnen Seiten	2-14
2.2.2.1	Namenspeicher	2-14
2.2.2.2	Triadenspeicher	2-16

2.2.2.3	Datenspeicher	2-25
2.3	<u>Die verschiedenen Ebenen des Systems</u>	2-29
2.3.1	Systemdienste und Hilfsroutinen	2-29
2.3.1.1	Aufteilung in Anwender-bezogene Gruppen (Ebenen) von Unterprogrammen	2-29
2.3.1.2	Systemdienste	2-32
2.3.1.3	Hilfsroutinen	2-33
2.3.1.4	Die Systemstruktur aus der Sicht des Anwenders	2-35
2.3.2	Die Verweisstruktur des Systems	2-36
2.3.2.1	Die Verweisebenen	2-36
2.3.2.2	Darstellung der Struktur	2-37
3.	<u>BESCHREIBUNG DER PROGRAMME</u>	3-1
3.1	<u>Vorbemerkung</u>	3-1
3.2	<u>Die Programme der Ebene 0 (Dialogsystem)</u>	3-1
3.2.1	Gruppe D1 : PROGRAM DIALØG	3-1
3.2.1.1	Beschreibung des Programms	3-2
3.2.1.2	Ablauf des Dialogs	3-4
3.2.1.3	Befehlsvorrat	3-5
3.2.1.3.1	Systembefehle	3-5
3.2.1.3.2	Store-Befehle	3-5
3.2.1.3.3	Declare-Befehle	3-6
3.2.1.3.4	Delete-Befehle	3-6
3.2.1.3.5	Change-Befehle	3-7
3.2.1.3.6	Read-Befehle	3-7
3.2.1.3.7	Find-Befehle	3-7
3.2.1.3.8	List-Befehl	3-8
3.2.1.3.9	Dump-Befehle	3-9
3.2.2	Gruppe D3	3-9
3.2.2.1	ALPHAN - Alpha Namen-Parameter	3-9
3.2.2.2	ALPHAD - Alpha Daten-Parameter	3-10
3.2.2.3	NUM - Numerische Parameter	3-12
3.3	<u>Die Programme der Ebene 1 (Unterprogramme)</u>	3-13
3.3.1	Gruppe G1 (Manipulation von Single-Value-Entities)	3-13
3.3.1.1	STSNGL - Store Single	3-13
3.3.1.2	CHSNGL - Change Single	3-14
3.3.1.3	DLSNGL - Delete Single	3-14

3.3.2	Gruppe G2 (Manipulation von Namen)	3-15
3.3.2.1	CHNAME - Change Name	3-15
3.3.2.2	DECSYN - Declare Synonym	3-16
3.3.2.3	DLNAME - Delete Name	3-17
3.3.3	Gruppe G3 (Manipulation von Mengen)	3-18
3.3.3.1	DECSET - Declare Set (Declare Class)	3-18
3.3.3.2	STELEM - Store Element	3-19
3.3.3.3	DLELEM - Delete Element	3-20
3.3.3.4	DLSET - Delete Set (Delete Class)	3-21
3.3.4	Gruppe G4 (Manipulation von Relationen)	3-23
3.3.4.1	DECREL - Declare Relation	3-23
3.3.4.2	STRELE - Store Relation Entry	3-24
3.3.4.3	DLRELE - Delete Relation Entry	3-28
3.3.5	Gruppe G5 (Lesen von Daten)	3-29
	RDENT - Read Entity	3-29
3.3.6	Gruppe G6 (Lesen von Namen)	3-30
3.3.6.1	RDNAME - Read Name	3-30
3.3.6.2	RDSYN - Read Synonym	3-32
3.3.7	Gruppe G7 (Lesen von Informationen über Mengen)	3-33
3.3.7.1	FDELEM - Find Element	3-33
3.3.7.2	LIELEM - List Element	3-35
3.3.8	Gruppe G8 (Lesen von Informationen über Relationen)	3-36
3.3.8.1	FDRELE - Find Relation Entry	3-36
3.3.8.2	FDRTOP - Find Ring-Top	3-40
3.3.8.3	FDNEXT - Find Next	3-41
3.3.9	Gruppe G9 (Systemprogramme)	3-43
3.3.9.1	SYSINT - System Initialisierung	3-43
3.3.9.2	PAGE - Seitentransport	3-44
3.3.9.3	SYSGCL - System Garbage Collection	3-45
3.3.9.4	SYSOUT - System out	3-46
3.4	<u>Die Systemdienste der Ebene 2</u>	3-47
3.4.1	Gruppe S1 (Manipulation von Entities)	3-47
3.4.1.1	STENT - Store Entity	3-47
3.4.1.2	DELENT - Delete Entity	3-52
3.4.1.3	DSHIFT - Data Shift	3-55
3.4.1.4	RTRNID - Return ID	3-56

3.4.2	Gruppe S2 (Manipulation von Triaden)	3-58
3.4.2.1	STØRE - Store Triade	3-58
3.4.2.2	DELETE - Delete Triade	3-61
3.4.2.3	RETRIV - Retrieve Triade	3-63
3.4.2.4	GARCØL - Garbage Collection im Triadenspeicher	3-66
3.4.3	Gruppe S3 (Speicherinitialisierung)	3-70
3.4.3.1	NSINIT - Namenspeicher-Initialisierung	3-70
3.4.3.2	F1INIT - Initialisierung des freien Speichers F1	3-71
3.4.3.3	HSINIT - Initialisierung der Hash-Area HAS	3-72
3.4.3.4	F2INIT - Initialisierung des freien Speichers F2	3-72
3.4.3.5	DSINIT - Datenspeicher-Initialisierung	3-73
3.4.4	Gruppe S4 (Ausgabe von Texten und Fragen)	3-74
3.4.4.1	ASK - Fragen an den Anwender	3-74
3.4.4.2	ERRØR - Fehlermeldungen und Fehlerstops	3-76
3.5	<u>Die Hilfsroutinen der Ebene 2</u>	3-77
3.5.1	Gruppe H1 (Programme, die mit Namen- und Daten- speicher arbeiten)	3-77
3.5.1.1	ALCNAM - Allocate Name	3-77
3.5.1.2	HASHNM - Hash Name	3-79
3.5.1.3	EXDATA - Extract Data	3-81
3.5.2	Gruppe H2 (Programme, die mit dem Triadenspeicher arbeiten)	3-82
3.5.2.1	KEEPFR - Keep Free	3-82
3.5.2.2	AALLØC - A-Allocate	3-83
3.5.2.3	REQCL1 - Request Class 1	3-84
3.5.2.4	PUTRIE - Put Ring Into Entity	3-86
3.5.2.5	BRSTRT - B-Ring-Start	3-89
3.5.2.6	BKTIE - Back Tie	3-89
3.5.2.7	F2REL - F2-Release	3-90
3.5.2.8	BRETRI - B-Retrieval	3-91
3.5.2.9	CRETRI - C-Retrieval	3-93
3.5.2.10	ALØØK - A-Look	3-95
3.5.2.11	GØBØTM - Go to Bottom	3-96
3.5.3	Gruppe H3 (Programme ohne Speicherbezug)	3-96
3.5.3.1	PEØRDR - Permutation Order of Request	3-96
3.5.3.2	IHASH - Hash Triaden	3-97
3.5.3.3	DINCØD - Data In Coding	3-99
3.5.3.4	DØUTCD - Data Out Coding	3-101

3.5.4	Gruppe H4 (Dump-Programme)	3-103
3.5.4.1	DUMPN - Dump des Namenspeichers	3-103
3.5.4.2	DUMPE - Dump des Entityspeichers	3-103
3.5.4.3	DUMP - Dump des Triadenspeichers	3-104
3.6	<u>Voraussetzungen über den Inhalt der FØRTRAN-Bibliothek</u>	3-105
	Gruppe FB (Funktionen in der FØRTRAN-Bibliothek)	
	IEØR - Exclusive Or	3-105
4.	<u>HINWEISE FÜR DEN GEBRAUCH DES SYSTEMS</u>	4-1
4.1	<u>Ausgegebene Meldungen</u>	4-1
4.1.1	Fehlerstops	4-1
4.1.2	Systemmeldungen	4-1
4.1.3	Aufforderungen an den Dialog-Partner	4-3
4.1.4	Meldungen über Entities	4-4
4.1.5	Sonstige Meldungen	4-4
4.2	<u>Technische Voraussetzungen für den Einsatz von DATAS</u>	4-4
4.2.1	Speicherplatz	4-4
4.2.2	Wortlänge	4-5
4.2.3	Externe Geräte und Dateien (Files), Ein- und Ausgabe	4-6
5.	<u>VERGLEICH MIT DEM URSPRÜNGLICHEN KONZEPT</u>	5-1
5.1	<u>Definition des ursprünglichen Konzepts von DATAS</u>	5-1
5.2	<u>Entities</u>	5-2
5.2.1	Wortlänge und IDs	5-2
5.2.2	Entity-Klassen	5-3
5.2.3	Adressierung der Entities	5-3
5.3	<u>Triaden</u>	5-4
5.3.1	Einteilung der Seiten des Triadenspeichers	5-4
5.3.2	Struktur der A-, B- und C-Zellen	5-4
5.3.3	Bit-Manipulationen	5-5
5.4	<u>Programme</u>	5-5
5.4.1	Triaden-Programme	5-5
5.4.2	Entity-Programme	5-6
5.5	<u>Zusammenfassung</u>	5-6

A-1

A1	Information Retrieval mit dem Dialogsystem : Stammbaum	A-1
A2	Einsatz des Dialogsystems zur Bearbeitung graphischer Daten	A-8
A3	Einsatz von DATAS im Stapelbetrieb : Rückkopplungen in einem JK-Flipflop	A-1

Literaturverzeichnis L-1

0. Vorwort

DATAS (Datenstrukturen in Assoziativer Speicherung) ist eine assoziative Datenstruktur. Die Entwicklung eines implementierungsfähigen Konzepts der Struktur wurde als Diplomarbeit für Herrn Dipl.-Ing. V. Glatzer vergeben. Die Diplomarbeit erfolgte 1971 am Lehrstuhl und Institut für Informationsverarbeitung der Technischen Universität Berlin. In Zusammenarbeit der Forschungsgruppe Rechnerorganisation und Computer Graphics mit der Forschungsgruppe Schaltkreise und Schaltwerke im Fachbereich Angewandte Mathematik und Informatik der Universität des Saarlandes wurde die Diplomarbeit von Herrn Glatzer überarbeitet. Die dabei entstandene neue, leicht abgeänderte Version von DATAS wurde von Herrn Dipl.-Phys. G. Weck für die Telefunken TR440 implementiert. Es ist vorgesehen, diese Implementierung im Lehr- und Forschungsbetrieb einzusetzen und dabei statistisches Material über die Bedingungen dieses Einsatzes und die Flexibilität der vorliegenden Implementierung zu gewinnen. Weiterhin ist geplant, DATAS als Simulationsmodell für andere Konzepte von Datenstrukturen einzusetzen.

Die Autoren möchten den Mitarbeitern des Rechenzentrums der Universität des Saarlandes ihren Dank für die weitgehende Unterstützung bei der Implementierungsarbeit aussprechen. Hier ist insbesondere Herr B. Kett hervorzuheben, dessen Rat bei der Behebung von Schwierigkeiten und beim Suchen von Fehlern im Programm unentbehrlich war. Nicht zuletzt möchten sich die Autoren bei Herrn Prof. Dr. W. Giloi für das rege Interesse an diesem Projekt und für die Anregung zu dem vorliegenden Bericht herzlich bedanken.

1. Datenstrukturen in assoziativer Speicherung

1.1. Datenstrukturen

1.1.1 Allgemeines

Die in einem Rechner abgespeicherten Daten können nicht wahllos im Kernspeicher und/oder auf einem externen Speichermedium abgespeichert sein, da der Rechner in diesem Falle entweder für jeden einzelnen Datenzugriff den ganzen Speicher durchsuchen müßte oder in geordneter Form Verweise auf jedes einzelne Datenelement abgespeichert haben müßte. Das erste Verfahren ist aufgrund des bei heutigen Rechnern dafür notwendigen erheblichen Zeitaufwands untragbar. Das zweite Verfahren verlagert nur das Problem, denn hier brauchen zwar nicht die Daten strukturiert zu werden, doch die Verweislisten müssen eine Struktur besitzen, weil sonst sie ganz durchsucht werden müßten, und diese könnten im Extremfall genauso umfangreich wie die Daten werden / 1 /, / 2 /.

Aus diesen Gründen werden in allen Rechnern sämtliche Daten nach bestimmten, rechnerabhängigen Konventionen abgespeichert, d.h. sie werden irgendwie strukturiert. Dabei bestimmt die Lage eines Datenelementes mit, wie es zu interpretieren ist, und umgekehrt werden Datenelemente bestimmter Bedeutung an bestimmten, entsprechenden Stellen abgespeichert. Die Daten beispielsweise, die von einem Programm zu verarbeiten sind - letztlich gibt es ja in einem Rechner strenggenommen nur Daten - sind auch strukturiert. So wird für jede Variable festgelegt, ob sie eine Gleitkommazahl oder eine Festkommazahl darstellt oder ob sie als logische Variable oder als Zeichenkette aufzufassen ist. Die Menge der Variablen kann in Felder gegebener Dimensionen eingeteilt werden, und sie kann zu Files einer bestimmten Struktur zusammengefaßt werden.

Alle diese Strukturen bieten den großen Vorteil, daß sich Datenelemente anhand der Strukturinformation ohne großen Aufwand und ohne langes Suchen finden lassen, und daß ganze Mengen von Daten ähnlicher oder gleicher Bedeutung zu Begriffen zusammengefaßt werden können. Durch die Strukturierung der Daten ist es möglich,

durch Angabe der Begriffe dann jeweils die gesamte Datenmenge gleich zu manipulieren, ohne dabei auf jedes einzelne Datenelement Bezug nehmen zu müssen.

1.1.2 Assoziative Datenstrukturen

Die im vorigen Abschnitt genannten Datenstrukturen sind ausgezeichnet zur Behandlung numerischer Daten geeignet, können jedoch eine große Klasse von Problemen nur sehr umständlich und mühevoll bearbeiten. Zu dieser Klasse gehören Probleme des Information Retrieval, der Verarbeitung graphischer Daten, etc. Diese Probleme sind nicht-numerisch oder enthalten zum Mindesten eine starke nicht-numerische Komponente, die hauptsächlich aus dem Aufbau und der Beschreibung von Beziehungen (Assoziationen) zwischen den Datenelementen besteht. Man könnte daher diese Probleme assoziative Probleme nennen, und eine Datenstruktur, die versucht, solche Probleme lösbar zu machen, wird entsprechend auch als assoziative Datenstruktur bezeichnet. Es wird Ziel der folgenden Abschnitte sein, diesen zunächst noch etwas verschwommen und unexakt angegebenen Begriff zu präzisieren und die Begriffe, die zur Beschreibung einer solchen assoziativen Datenstruktur notwendig sind, klar herauszuarbeiten.

1.2 Adressierungsmöglichkeiten von Daten

Der folgende Abschnitt lehnt sich im Aufbau an die entsprechenden Abschnitte der Beschreibung des LEAP-Systems / 3 / an.

1.2.1 Explizite (direkte) Adressierung

Normalerweise werden die im Speicher (Kernspeicher oder externes Speichermedium) eines Rechners abgespeicherten Daten durch explizite Adressierung angesprochen. Darunter ist folgendes zu verstehen: Eine Information fester Länge (eine bestimmte Anzahl von Bits oder Worten) wird vom Programm aus durch einen Namen gekennzeichnet. Dieser Name wird nach bestimmten Vorschriften auf eine Zahl, die Adresse der Information, abgebildet. Bei der Ausführung des Programms wird dann nur noch mit dieser Adresse,

die den Ort der Information im Speicher angibt, gearbeitet. Das Programm kann also direkt auf die benötigte Information zugreifen.

1.2.2 Zeiger (Indirekte Adressierung)

Das einfache Konzept der direkten Adressierung gewährleistet zwar die größtmögliche Schnelligkeit beim Aufsuchen der Information, ist jedoch für viele Zwecke zu starr und zu wenig flexibel, da es für jede benötigte Information voraussetzt, daß ihre Adresse direkt bekannt ist. Das ist aber schon in vielen sehr einfachen Fällen nicht mehr gewährleistet, so zum Beispiel bei der Bestimmung des i -ten Elementes eines Feldes, wobei i eine Variable ist. Die Adresse dieses Elementes kann erst dann bestimmt werden, wenn außer der (bekannten) Anfangsadresse des Feldes auch der Wert der Variablen i bekannt ist, der seinerseits über eine weitere Adresse gefunden werden muß. Man hat hier den Fall, daß die unter einer Adresse gefundenen Daten nicht die gesuchte Information sondern erst deren Adresse enthalten. Diese Adressverweise können sich im Prinzip über beliebig viele Stufen erstrecken, sind jedoch bei vielen Maschinen aus technischen Gründen auf Hardware-Ebene auf eine bestimmte Anzahl von Stufen beschränkt. Auf der Ebene von Programmen, also auf der Software-Ebene, läßt sich dieses Verfahren ebenfalls nachbilden und ist dann nicht auf eine bestimmte Anzahl von Stufen festgelegt. Man nennt dieses Verfahren auf Hardware-Ebene oder bei direkter Benutzung der entsprechenden Hardware-Befehle von Programmen aus indirekte Adressierung. Die Nachbildung dieser indirekten Adressierung auf der Ebene von Programmen oder von höheren Programmiersprachen wird jedoch meist als Verarbeitung von Listen-, Ketten- oder Ringstrukturen bezeichnet. Solche Kettenstrukturen werden im Abschnitt 2.2.2.1 besprochen, Ringstrukturen im Abschnitt 2.2.2.2. Der Verweis von einem Element auf das nächste in einer solchen Struktur wird als Zeiger bezeichnet.

1.2.3 Hash-Codierung

Beim Abspeichern großer Mengen von Einzeldaten, die nicht in Form von Feldern oder ähnlichen Objekten vorstrukturiert sind, versagen die Methoden der direkten und der indirekten Adressierung. Eine direkte Adressierung ist unrationell, weil die Listen, die Namen und Adressen einander zuordnen, zu umfangreich und ihr Durchsuchen zum Finden einer bestimmten Adresse zu langwierig würde. Eine Anordnung der Daten selbst in einer Listen- oder Ringstruktur würde zwar das umfangreiche Adreßbuch der direkten Adressierung überflüssig machen, aber die Suchvorgänge nach einer bestimmten Information wären aufgrund der vielen Stufen indirekter Adressierung mindestens ebenso langwierig. Man kann sich hier jedoch durch eine Abbildungsvorschrift helfen, die direkt aus der abzuspeichernden Information die Adresse berechnet, an der sie abzuspeichern ist. Dieses Verfahren hat jedoch den Nachteil, daß man die Information, die man sucht, kennen muß, daß man also nur überprüfen kann, ob eine bestimmte Information vorliegt oder nicht.

Unterscheidet man jedoch zwischen zwei Teilen der Information, einem, der zum Suchen benötigt wird - z.B. dem Namen einer Variablen - und einem, der nicht benötigt, sondern gesucht wird - z.B. dem Wert der Variablen -, dann läßt sich dieses Problem lösen. Man erzeugt mittels einer Abbildungsvorschrift, des sogenannten Hash-Algorithmus, aus dem bekannten Teil der Information, dem Namen, ihre Adresse und speichert sie dort ab bzw. sucht dort den unbekannten Teil, den Wert. Aber auch diese Hash-Adressierung hat einen Nachteil. Im allgemeinen ist die Information, die im Namen enthalten ist, so umfangreich, daß die Anzahl der prinzipiell möglichen Namen bei weitem größer ist als die Anzahl der zur Verfügung stehenden Speicherplätze. Glücklicherweise ist jedoch die Anzahl der wirklich vorkommenden Namen erheblich geringer als die Anzahl der prinzipiell möglichen. Man denke dazu etwa an die Namen in einem Telefonbuch und vergleiche ihre Anzahl mit der aller Buchstabenkombinationen von, sagen wir, maximal 10 Buchstaben. Dieses Beispiel zeigt aber auch noch, daß es kein allgemeines Verfahren gibt, die für eine bestimmte Datenmenge sinnvollen Namen von den nicht vorkommenden,

sinnlosen Namen zu unterscheiden - im Extremfall muß für jede einzelne Buchstabenkombination (oder allgemeiner jede Kombination von Zeichen) explizit festgestellt werden, ob sie als sinnvoll oder sinnlos zu betrachten ist. Da andererseits aber der Bildbereich des Hash-Algorithmus in jedem Fall ein ähnlicher ist, nämlich eine endliche Untermenge der natürlichen Zahlen, folgt daraus, daß es keinen allgemein gültigen Hash-Algorithmus gibt, der in jedem Fall als der günstigste anzusehen wäre.

Da nämlich im allgemeinen nicht im voraus entschieden werden kann, welche Namen als sinnvoll, d.h. als zur Datenmenge gehörig, aufzufassen sind, muß als Urbild-Bereich der durch den Hash-Algorithmus bestimmten Abbildung die Menge der prinzipiell möglichen Namen gewählt werden. Nun ist deren Anzahl aber größer als die der zur Verfügung stehenden Speicherplätze, so daß die Abbildung mit Sicherheit nicht injektiv ist und immer eine bestimmte Anzahl prinzipiell möglicher Namen auf dieselbe Adresse abbildet. Zu jeder Adresse existiert somit eine Äquivalenzklasse auf sie abgebildeter tatsächlich vorkommender, also sinnvoller Namen. Im Gegensatz zu der Anzahl der auf diese Adresse abgebildeten prinzipiell möglichen Namen kann der Umfang der genannten Äquivalenzklasse nicht ohne die Kenntnis aller sinnvollen Namen oder eines Kriteriums, das für jeden Namen zweifelsfrei zu entscheiden gestattet, ob er sinnvoll oder nicht ist, bestimmt werden. Da nun im Interesse einer geringen Suchzeit vom Hash-Algorithmus verlangt werden muß, daß die Anzahl der Äquivalenzklassen mit mehr als einem Element möglichst gering und daß ihr Umfang ebenfalls möglichst klein sein muß, und da weiterhin im Interesse einer guten Speicherausnutzung verlangt werden muß, daß möglichst wenige Äquivalenzklassen leer sind, folgt, daß ohne Kenntnis aller abzuspeichernden Daten prinzipiell kein optimaler Hash-Algorithmus angegeben werden kann. Wenn man wie in den meisten tatsächlich vorkommenden Fällen lediglich die Menge aller prinzipiell möglichen Namen kennt und bestenfalls Vermutungen haben kann, welche dieser Namen in einem bestimmten Fall als sinnvoll bezeichnet werden sollen, läßt sich der Hash-Algorithmus nur auf gut Glück wählen, und erst der Anwender des Programms kann - im nachhinein - feststellen, ob für sein spezielles Problem der Hash-Algorithmus gut ist.

Man nennt den Fall, daß eine der Äquivalenzklassen mehr als nur ein Element besitzt, in Anlehnung an den englischen Sprachgebrauch eine Conflict-Situation. Über deren Behandlung bei den hier verwendeten Hash-Algorithmen wird ausführlich in den Abschnitten 2.2.2.1 und 2.2.2.2 gesprochen, die Hash-Algorithmen selbst werden in den Abschnitten 3.5.1.2 und 3.5.3.2 angegeben. (Vergleiche hierzu auch / 4 / und / 5 /).

1.2.4 Implizite (assoziative) Adressierung

Das Verfahren der impliziten Adressierung kann als Erweiterung der expliziten und der Hash-Adressierung aufgefaßt werden. Wie dort muß ein Teil der gesuchten Information bekannt sein, und dieser Teil wird ausgenutzt, um die Information im Speicher zu adressieren. Zusätzliche Adresseninformationen oder Algorithmen werden jedoch nicht mehr benötigt. Stattdessen wird die gesuchte Information als aus einzelnen Teilen bestehend aufgefaßt, die miteinander assoziiert sind. Man verwendet somit die Bedeutung der Variablen - ihren Zusammenhang mit anderen Variablen - zum Adressieren. Im Gegensatz zum explizit adressierbaren Speicher, dessen wesentlicher Inhalt die explizit gespeicherten Daten sind, werden hier primär die Beziehungen zwischen den Daten abgespeichert.

Da es zur Zeit noch keine größeren Hardware-Assoziativspeicher in den üblichen Rechnern gibt, muß die implizite Adressierung in einem normalen explizit adressierten Speicher simuliert werden. Daß dies prinzipiell möglich ist, sieht man leicht daran, daß man jede Assoziation durch sequentielles Durchsuchen des ganzen Speichers notwendig finden muß. Dieses Verfahren ist natürlich bei einigermaßen großen Speichern praktisch indiskutabel, da es viel zu viel Zeit kostet. Das Hauptproblem beim Aufbau eines Assoziativspeichers ist somit, eine Art der Abspeicherung und eine Suchstrategie zu finden, die einen möglichst schnellen Zugriff auf die Informationen des Speichers gestatten. Hierzu wurden inzwischen verschiedene Möglichkeiten vorgeschlagen und zum Teil auch implementiert. In Anlehnung an / 6 / sollen hier kurz die assoziativen Datenstrukturen CØRAL, LEAP und SAM mit der in diesem Bericht beschriebenen Struktur

von DATAS verglichen werden.

1.3 Merkmale und Vergleich verschiedener assoziativer Datenstrukturen

1.3.1 CØRAL

Struktur- und Dateninformation werden physikalisch zu Blöcken zusammengefaßt abgespeichert. Dabei werden die Daten der Struktur durch Erzeugen eines Blocks fester Länge, dessen Form von der Klasse der Daten bestimmt wird, hinzugefügt. Für jede Klasse gibt es einen vorspezifizierten Prototyp, so daß alle Eigenschaften, die auf Größen einer Klasse anzuwenden sein werden, vor der Exekution bekannt sein müssen. Für die Spezifikation aller Eigenschaften muß Platz bereitgehalten werden, so daß bei umfangreicheren Assoziativstrukturen sehr schnell die Grenzen des zur Verfügung stehenden Speicherplatzes erreicht werden. Die Strukturinformation wird dabei in einem umfangreichen Ring-system abgespeichert. Da die umfangreichen Datenblöcke zusammen mit der Struktur abgespeichert werden, müßten bei Anschluß eines Paging-Systems bei Absuchen eines Ringes, der die Strukturinformation enthält, sehr viele Sekundärspeicherzugriffe erfolgen. Eine dynamische Erweiterung der Struktur um neue Assoziationen ist ohne Reorganisation der gesamten Struktur nicht möglich.

Die genannten Eigenschaften zeigen, daß CØRAL für umfangreiche und dynamisch verändernde Datenstrukturen ungünstig ist. Vorteilhaft ist dieses System jedoch für Anwendungen, in denen jedes Datenelement mit vielen anderen Datenelementen assoziiert ist und für Anwendungen, in denen oft Ringe rückwärts abzusuchen sind, was durch die in jedes zweite Ringelement eingebauten Rückwärtszeiger erheblich erleichtert wird (Vergleiche hierzu / 7 /.) .

1.3.2 LEAP

Hier werden logische Tatbestände in einer Form

"Attribute" of "Object" is "Value"

abgespeichert. Ein solcher Tatbestand, der in der genannten Art und Weise als TRIPLE von ITEMS ("Attribute", "Object", "Value") dargestellt wird, wird als FACT bezeichnet. Diese FACTs sind nur implizit in der Struktur vorhanden, sie können ebenso wie die Attributes, die Beziehungen zwischen den ITEMS darstellen, nicht wie ITEMS behandelt werden. Diese stellen die kleinste direkt adressierbare Informationseinheit dar, der ein interner Schlüssel und ein Wert zugeordnet sind, wobei der Wert von einem Datentyp der Grundsprache sein muß (REAL, INTEGER, usw.) Die statische und dynamische Speicherung von ITEMS ist möglich, ebenso der Anschluß eines Paging-Systems, das es ermöglicht, große Datenmengen, die auf einem externen Speichermedium stehen, zu verarbeiten. Die Speicherstruktur ist in drei Listen (A-, O- und V-Liste) unterteilt, die als umfangreiche Ringsysteme organisiert sind. Von jedem Listentyp muß sich jeweils eine Seite im Kernspeicher befinden. Auf einen bestimmten Teil des Ringsystems kann, wie auch bei DATAS, mittels eines Hash-Algorithmus zugegriffen werden. (Vergleiche hierzu auch / 3 / und / 8 /.)

1.3.3 SAM

Die logische Information wird hier durch TRIADEN ABC dargestellt, die ebenfalls die Beziehung "Attribute" of "Object" is "Value" repräsentieren und mathematisch binäre Relationen

$$A(B,C) , A \in A' , B \in B'$$

darstellen. Jede der auftretenden Größen (Objekte und Relationen) wird im sogenannten DATA SPACE durch eine ENTITY dargestellt, die ihrerseits Element einer Klasse ist, die eine Menge mit gleichen Eigenschaften spezifiziert. Da Relationen hier ebenfalls als Entities abgespeichert werden, lassen sich auch Relationen über Relationen definieren.

Im Unterschied zu LEAP ist die kleinste auffindbare Informationseinheit hier das DATA-ITEM, das sich im Datensatz einer Entity befindet und dem ein Schlüssel und ein Wert zugeordnet sind. Die kleinste direkt adressierbare Informationseinheit ist dagegen die Entity, der kein Wert, sondern eine Menge von Data-Items zugeordnet ist. Interpretiert bzw. spezifiziert werden diese Data-Items einer Entity durch die übergeordnete Klassen-Entity, deren Datensatz wiederum auch von einer übergeordneten Entity, der standard definition, spezifiziert wird. Die Permutationen eines Triples, also eine Triade, werden in einem Triadenspeicher gespeichert. Die Speicherung von Entities kann vom Benutzer so gesteuert werden, daß logisch verwandte Größen auch physikalisch benachbart abgespeichert werden können, ein Verfahren, das aufgrund der sequentiellen Indexstrategie von DATAS auch dort möglich ist. Das Paging-System von SAM ist außerdem flexibler als das von LEAP, da hier für den Überlauf ein "Overflow-Pool" von Seiten zur Verfügung steht./9/,/10/

1.3.4 DATAS

Das in dieser Arbeit beschriebene DATAS-System wurde aus LEAP und SAM entwickelt. Es übernimmt die in LEAP verwendeten Ringstrukturen zur Beschreibung der Triples, ebenso die Möglichkeit der Hash-Adressierung im Triadenspeicher. Von SAM ist die Abspeicherung aller Permutationen eines Triples auf derselben Seite des Triadenspeichers und die Behandlung auch von Relationen als Entities übernommen worden. Dagegen konnte durch die formatfreie Abspeicherung der Entities auf die Einführung von Klassen-Entities und Standard-Definitionen verzichtet werden. Eine Abspeicherung von Mengen und Elementbeziehungen wurde dennoch durch die Einführung von Multi-Value-Entities möglich. Ein Paging-System wurde vorgesehen, allerdings zur Zeit noch ohne Vorkehrungen für den Fall eines Überlaufs einer Seite. Eine Erweiterung des Paging-Systems in dieser Richtung ist jedoch jederzeit möglich. Das System sieht außerdem eine Adressierung der Entities über jeweils mehrere Namen (Synonyme) vor.

1.4 Beschreibung des DATAS-Systems

Die folgende Beschreibung lehnt sich in ihrem Aufbau an / 11 / an.

1.4.1 Die Komponenten des Systems

Die Basiselemente der hier vorgestellten Implementierung von DATAS (DATenstrukturen in ASsoziativer SPeicherung) sind :

Namen

Entities

Triaden .

Dabei werden **Entities** auf Benutzer-Ebene über Namen (Zeichenketten beliebiger Länge, deren erste 10 Zeichen unterschieden werden) adressiert; dabei kann dieselbe Entity verschiedene Namen (Synonyme) haben. Jede Entity kann Daten enthalten, deren Länge zur Zeit auf 800 Zeichen (400 Maschinenworte) beschränkt ist. Diese Einschränkung ist willkürlich und kann jederzeit vom Anwender geändert werden. Intern werden diese Entities direkt adressiert, ihre Adressen werden als IDs bezeichnet. Triaden sind Tripel solcher IDs, sie werden zur Darstellung von Relationszusammenhängen benutzt.

Die in der Struktur enthaltenen Informationen werden in drei Speicherbereichen abgespeichert.

Namenspeicher (Zuordnung von IDs zu Namen)

Triadenspeicher (Relationszusammenhänge, dargestellt durch Triaden)

Datenspeicher (Entities)

Jeder dieser Speicher ist in Seiten von je 2048 Worten unterteilt; diese Seiten stehen auf einem externen Speichermedium, und jeweils eine Seite von jedem Bereich muß im Kernspeicher gehalten werden.

Das System besteht aus drei Ebenen von Programmen :

Dialogprogramm

Unterprogramme (vom Anwender aufzurufen)

Systemdienste und Hilfsroutinen (für den Anwender verborgen)

Die Unterprogramme sehen Möglichkeiten vor zur Manipulation von Daten, Namen, Mengen und Relationen, zum Suchen von Informationen über diese Objekte und zur Initialisierung und Maintenance der Strukturen sowie zum Seitentransport. Diese Unterprogramme können vom Dialogprogramm oder jedem beliebigen Anwenderprogramm im Stapel- oder Dialogbetrieb, das den Aufruf von FØRTRAN IV - SUBRØUTINES gestattet, aufgerufen werden. Das Dialogprogramm gestattet auch einen direkten Aufruf mittels vom Terminal eingegebener Befehle; es arbeitet als Interpreter. Systemdienste und Hilfsroutinen behandeln die im System vorhandenen Größen (Namen, Entities und Triaden) direkt. Sie können (oder sollten) nicht direkt vom Anwender aus aufgerufen werden, sondern nur indirekt über die Unterprogramme der mittleren Ebene. Alle Programme stehen in FØRTRAN IV als Quellprogramm und für die Telefunken TR 440 auch als Bibliothek übersetzter Programme und als lauffähiges, fertig montiertes Dialogsystem zur Verfügung. Eine genaue Beschreibung der vom System verlangten Betriebsmittel und der bei seinem Einsatz zu beachtenden Überlegungen befindet sich im Abschnitt 4.2 dieses Berichts.

1.4.2 Entities

Es sind drei Klassen von Entities vorgesehen:

- Relation-Entities
- Single-Value-Entities
- Multi-Value-Entities

Relation-Entities repräsentieren Beziehungen zwischen Entities; diese Beziehungen werden durch die in Abschnitt 1.4.3 beschriebenen Triaden repräsentiert. Single-Value-Entities sind einfache Datenelemente; sie können als "Object", "Value" oder auch als Elemente von Mengen auftreten. Multi-Value-Entities repräsentieren Mengen. Eine Entity kann nicht zugleich mehreren Klassen angehören, jedoch wird die Klasse von Single-Value-Entities, die als Relation oder als Menge angesprochen werden, automatisch umgeändert. Eine Umänderung zwischen Relation und Menge ist jedoch nicht vorgesehen.

Jede Entity kann über praktisch beliebig viele Namen (Synonyme) adressiert werden und kann beliebige Daten enthalten. Zusätzlich enthalten Multi-Value-Entities die IDs ihrer Elemente und Relation-Entities Information über die Verteilung der Relation auf die verschiedenen Seiten des Triadenspeichers.

1.4.3 Triaden

Die logischen Beziehungen zwischen den Entities werden als Elemente von zweistelligen (binären) Relationen betrachtet und in Form von geordneten Tripeln, den Triaden ABC, abgespeichert.

$$A \subseteq B' \times C' \quad , \quad A = \{(B,C) \mid B \in B' \quad , \quad C \in C' \quad , \quad H(B,C)\} =: \bigcup_{\substack{B \in B' \\ C \in C' \\ H(B,C)}} \{(B,C)\}$$

Die Relation A ist eine Menge von geordneten Paaren (B,C), für die eine Aussage H(B,C) gilt. B' und C' sind die Basismengen von A, sie können gleich der Menge E aller Entities angenommen werden. Damit ergibt sich die in Abschnitt 2.1.3 verwendete Beschreibung der Relationen und Triaden. Jeder Relation wird ein Permutationstyp (0,1 oder 2) zugeordnet, der bestimmt, welche assoziativen Fragen zu dieser Relation beantwortet werden können. Bei Relationen der Klasse 0 können die Fragetypen

FO(ABC) : existiert der Zusammenhang ABC, d.h. $\exists (B,C)$?

F1(AB?) : welche Entity (oder Entities) steht (stehen) zu B in der Relation A ?

F4(A??) : gibt es Entities, die in der Relation A zueinander stehen ?

beantwortet werden. Bei Relationen der Klasse 1 können zusätzlich die Fragetypen

F3(?BC) : in welcher Relation stehen B und C zueinander ?

F6(?B?) : gibt es eine Relation, in der irgendeine Entity zu B steht ?

beantwortet werden. Bei Relationen der Klasse 2 sind schließlich auch noch die verbleibenden Fragetypen

F2(A?C) : zu welcher Entity (zu welchen Entities) steht C in der Relation A?

F5(??C) : gibt es eine Relation, in der C zu irgendeiner Entity steht?

beantwortbar. Der Permutationstyp und damit die Art der erlaubten Fragen sind für alle Triaden derselben Relation gleich, können aber bei verschiedenen Relationen verschieden angegeben werden. Die oben genannten 7 Fragetypen lassen sich leicht in drei Klassen einteilen :

Klasse 0 : FO : gilt ein bestimmter, angegebener Relationszusammenhang ?

Klasse 1 : F1,F2,F3 : welche Entity (Entities) steht (stehen) zu zwei vorgegebenen Entities in einem Relationszusammenhang ?

Klasse 2 : F4,F5,F6 : gibt es Entities, die zu einer vorgegebenen Entity in einem Relationszusammenhang stehen ?

DATAS gestattet die Beantwortung aller 3 Frageklassen. Bei Fragen der Klasse 0 erhält man als Antwort die Feststellung, ob der genannte Relationszusammenhang besteht. Die Antwort auf Fragen der Klasse 1 ist verschieden, je nachdem, ob eine oder mehrere Entities gefunden wurden, die den gewünschten Relationszusammenhang erfüllen. Im ersten Falle erhält man als Antwort die gefundene Entity; im zweiten Falle wird eine Multi-Value-Entity als eine Art Sammelbegriff erzeugt, die als ihre Elemente alle gefundenen Entities enthält. Bei Fragen der Klasse 2 erhält man schließlich genügend Information über die Ringstrukturen, die die angegebene Entity enthalten, um mittels weiterer Fragen der Klasse 1 detaillierte Informationen einholen zu können.

1.4.4 Beispiel einer Assoziation in DATAS

Gegeben sei eine Relation

$$A = \{(B,C) \mid B \in \text{"Fahrzeuge"}, C \in \text{"Flächen"}, H(B,C) : \text{"C ist eine Fläche von B"}\}.$$

Ihre Elemente seien durch folgende Terme spezifiziert :

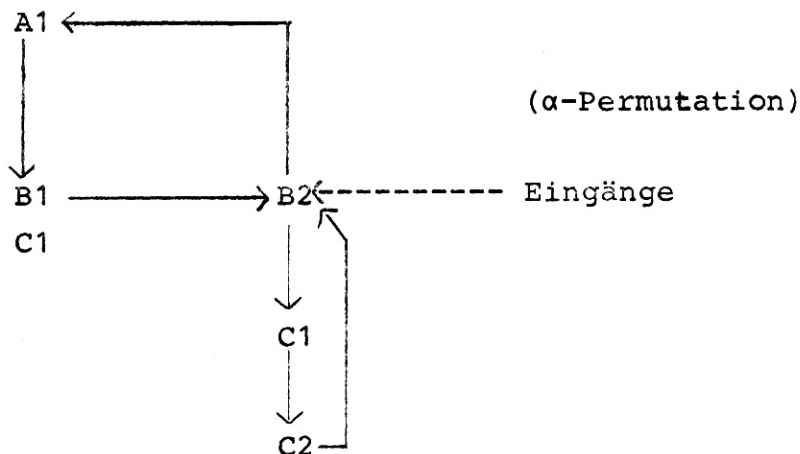
$$\begin{aligned} A1 &= \{(B1, C1), (B2, C1), (B2, C2)\} \\ &= \{(\text{Auto}, \text{Tür}), (\text{Bus}, \text{Tür}), (\text{Bus}, \text{Dach})\} \end{aligned}$$

Dabei können, im Gegensatz zu SAM und zum ursprünglichen Konzept von DATAS, die Basismengen "Fahrzeuge" und "Flächen" durch die Menge aller Entities ersetzt werden, so daß auch Bildungen wie $(A1, B1)$ als Elemente von $A1$ erlaubt sind.

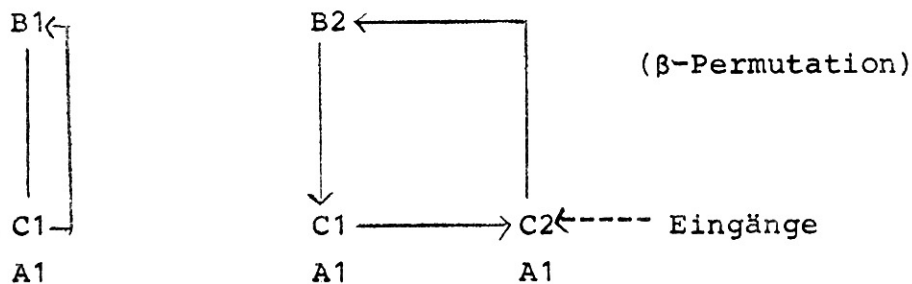
Diese Relation wird im DATAS-System in der Form

$$A1 = \{(B1, C1), (B2, \{C1, C2\})\}$$

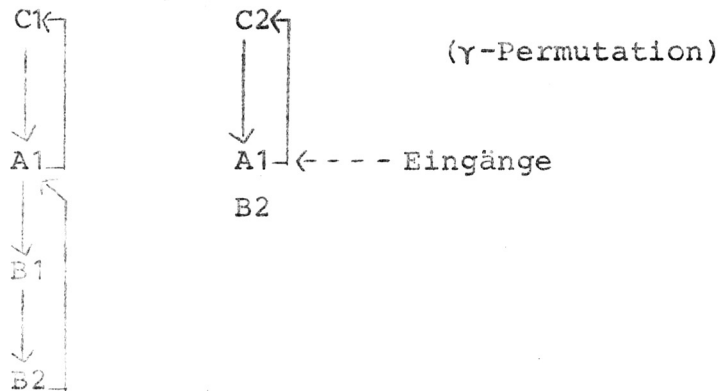
durch Zusammenfassen gleicher erster Komponenten von Paaren gespeichert. Die im Triadenspeicher aufgebaute Struktur bietet folgendes Bild (beim Permutationstyp 0) :



Beim Permutationstyp 1 wird zusätzlich folgende Struktur abgespeichert :



Schließlich wird beim Permutationstyp 2 als dritte Struktur die folgende abgespeichert :



Als Eingänge werden immer die Teile der Struktur bezeichnet, die in der Mitte liegen und zwei Ringen gleichzeitig angehören können. Sie werden nachher, in Anlehnung an die oben gewählte Bezeichnung bei der α-Permutation, als B-Zellen bezeichnet, die anderen Teile der Struktur entsprechend als A-Zellen bzw. C-Zellen. Die Bezeichnung "Eingänge" soll ausdrücken, daß man zu diesen Zellen beim Retrievalvorgang direkten Zugriff hat. Im Gegensatz dazu werden sowohl die Elemente des Vorbereiches der Relation als auch die ihres Nachbereiches im folgenden Entries genannt; die Begriffe "Eingang" und "Entry" dürfen somit nicht verwechselt werden.

Wenn man vom System nun Information über Fahrzeugflächen erhalten will, kann man die Frage (A1 ? ?) stellen und erhält als Antwort die Adresse des Anfanges des zu A1 gehörenden B-Ringes. Nachdem man nun festgestellt hat, daß das zu dieser Adresse gehörende Element B1 (Auto) ist, kann man die Frage (A1 B1 ?) stellen und erhält die Antwort C1(Tür). Hat man jedoch den B-Ring weiter abgesucht, weil B1 eine unbefriedigende Antwort war, und B2(Bus) gefunden, so erhält man auf die Frage (A1 B2 ?) die Antwort *C. Dabei ist *C eine Menge, die die Elemente C1(Tür) und C2(Dach) hat und als der Sammelbegriff "Flächen eines Bus" aufgefaßt werden kann.

1.4.5 Ausblick

Es ist zur Zeit geplant, DATAS als Datenstruktur für Anwendungen auf dem Gebiet der Computer Graphics einzusetzen. Dazu muß an den Unterprogrammen kaum etwas oder gar nichts geändert werden; jedoch müssen völlig neue Programme geschrieben werden, die die DATAS-Unterprogramme im graphischen System einsetzen. Die hauptsächlichsten Änderungen, die am DATAS-System selbst im Laufe der nächsten Zeit durchgeführt werden, falls sich dazu die Notwendigkeit ergibt, sind vermutlich, neben der Korrektur von möglichen Fehlern, die bei den bisherigen Tests noch nicht gefunden wurden, die Einführung eines Overflow-Pools für das Paging-System, zunächst wahrscheinlich für den Namenspeicher, und eventuell eine Erweiterung der Anzahl der Unterprogramme und der vom Dialogsystem zu verarbeitenden Befehle nach Bedarf. Wenn sich die verwendeten Hash-Algorithmen oder die Strategien des Paging-Systems im Laufe des Betriebes von DATAS als ungeeignet erweisen sollten, werden sie gegen andere ausgetauscht, was ohne sonstige Eingriffe ins System geschehen kann. Nach den bisherigen Erfahrungen dürfte dies jedoch nicht nötig sein. Weiterhin ist für die nächste Zukunft geplant, mit dieser Implementierung statisches Material über den Einsatz assoziativer Datenstrukturen und dabei eventuell auftretende Engpässe zu sammeln. Ferner soll die vorliegende DATAS-Implementierung benutzt werden, um andere Assoziativ-Speicher-Konzepte mit verhältnismäßig geringem Programmieraufwand simulieren zu können.

2. Die Struktur des Systems

2.1 Aufteilung in Gruppen von Unterprogrammen

2.1.1 Forderungen an das System

Ein System zur Manipulation mengen- und relationstheoretisch definierter Begriffe muß verschiedenen und verschiedenartigen Forderungen genügen:

- (1) Die Datenelemente sollen die zu verarbeitenden Begriffe in irgendeiner Weise eindeutig repräsentieren.

Hierzu ist es notwendig, eine Zuordnungsfunktion:

Begriff $\rightarrow$ Datenelement zu definieren. Für den Anwender am bequemsten ist folgende Art der Zuordnung:

- (2) Die Datenelemente sollen vom Anwender über Zeichenkettenvariable adressiert werden.

Die Erfüllung dieser Forderung gestattet es dem Anwender, die ihm gewohnte Terminologie beizubehalten. Zusammen mit (1) ergibt sich daraus:

- (2.1) Die Zuordnung: Zeichenkettenvariable $\rightarrow$ Datenelement muß eindeutig sein.

Da der Anwender aber nicht nur Information in das System eingeben will, sondern auch wieder vom System Information erhalten will, läßt sich (2.1) verschärfen zu:

- (2.2) Die Zuordnung: Zeichenkettenvariable $\rightarrow$ Datenelement muß umkehrbar eindeutig sein.

Anmerkung: Aus Gründen der Speicherplatzersparnis ist es sinnvoll, eine maximale Länge für die verwendeten Zeichenketten (Character-Strings) anzugeben. Diese Länge wurde hier (willkürlich) gleich 10 Character (Zeichen) gewählt. Bei längeren Zeichenketten werden das elfte und alle folgenden

Zeichen ignoriert. (Siehe auch Abschnitt 3.3.1.1).

Da die Anzahl der so herstellbaren Zeichenketten größenordnungsmäßig 10^{16} ist, läßt sich der binäre Wert der Zeichenkette nicht direkt als Adresse verwenden.

- (3) Es muß eine Zuordnung: Zeichenkettenvariable $\longleftrightarrow$ Adresse der abgespeicherten Daten existieren.

Aus (2.1) folgt sofort:

- (3.1) Die Zuordnung (3) muß eindeutig sein.

Für den Anwender kann es von Nutzen sein, bestimmte Datenelemente unter verschiedenen Namen anzusprechen. Somit gilt:

- (3.2) Die Zuordnung (3) muß nicht umkehrbar sein.

Die Forderungen (2.2) und (3.2) stehen nicht zueinander im Widerspruch, wenn es zunächst auch so den Anschein hat.

(2.2) garantiert nämlich, daß der Anwender auf eine Frage an das System eine eindeutige und reproduzierbare Antwort erhält; (3.2) ermöglicht ihm, (bei anderer Fragestellung) alle Zeichenketten zu erhalten, die das Datenelement adressieren.

Unter dem Gesichtspunkt einer schnellen Manipulation der abzuspeichernden und der abgespeicherten Daten läßt sich weiterhin folgende Forderung erheben:

- (4) Die Datenelemente sollen vom System direkt durch ihre Adressen angesprochen werden.

Aus (2) und (4) erhält man die Folgerung:

- (4.1) Es müssen dem Anwender zwei Pakete von Programmen zur Verfügung gestellt werden:
1. Programme, die aus den Zeichenketten die Adressen erzeugen
 2. Programme, die aus den Adressen die zugehörigen Zeichenketten bestimmen.

Weiterhin soll das System in der Lage sein, Datenelemente zu erzeugen, die Mengen darstellen, und Mengenoperationen zwischen diesen Datenelementen auszuführen. Daraus ergeben

sich die Forderungen:

- (5) Der Anwender muß die Möglichkeit haben, Datenelemente zu Mengen zusammenzufassen und diese Mengen durch Datenelemente darzustellen.

Um mit diesen Mengen arbeiten zu können, benötigt der Anwender Programme, die Mengeninformation ins System eingeben, und Programme, die sie vom System erhalten können:

- (5.1) Es müssen dem Anwender zwei Pakete von Programmen zur Verfügung gestellt werden:
 - 1. Programme, die Information über Mengen ins System einbringen
 - 2. Programme, die Information über Mengen vom System erhalten.

Schließlich soll das System in der Lage sein, Datenelemente zu erzeugen, die binäre Relationen (Relationen mit zwei Eingängen) darstellen, und diese Reaktionen zu manipulieren.

Daraus ergeben sich die Folgerungen:

- (6) Der Anwender muß die Möglichkeit haben, Datenelemente zu erzeugen, die binäre Relationen zwischen anderen Datenelementen darstellen.

Damit der Anwender mit diesen Relationen arbeiten kann, müssen ihm Programme zur Verfügung gestellt werden, die Information über Relationen mit dem System austauschen können:

- (6.1) Es müssen dem Anwender zwei Pakete von Programmen zur Verfügung gestellt werden:
 - 1. Programme, die Information über Relationen ins System einbringen
 - 2. Programme, die Information über Relationen vom System erhalten.

Die folgenden Forderungen sind mehr technischer Art.

Die Zuordnung von Adressen zu Zeichenkettenvariablen, die Darstellung von Relations-Zusammenhängen und die Darstellung der

Daten setzt Speicherbereiche einer bestimmten Struktur voraus. Daraus ergibt sich die Forderung:

- (7) Dem Anwender muß ein Programm zur Verfügung gestellt werden, das die Strukturen strukturierter Speicherbereiche erzeugt.

Nach dem Löschen von Datenelementen ist die Struktur der abgespeicherten Daten möglicherweise lokal zerstört. Da es dem Anwender nur zuzumuten ist, dieser Tatsache auf der Ebene der von ihm definierten Begriffe Rechnung zu tragen, und da auch Versehen des Anwenders nicht auszuschließen sind, ergeben sich die Forderungen:

- (7.1) Löschmodulare sind so aufzubauen, daß auch nach einem Löschvorgang die Möglichkeit gewährleistet sein muß, Information mit dem System auszutauschen.
- (7.2) Sofern bei einem Löschvorgang Speicherbereiche entstehen, die nicht mehr vom Anwender ansprechbar sind, müssen Programme zur Verfügung gestellt werden, die es gestatten, diese Speicherbereiche wieder freizugeben.
- (7.3) Bei Löschvorgängen, die ein Versehen des Anwenders möglich erscheinen lassen, sollte vor der Löschung rückgefragt werden. (Dies ist allerdings nur bei Dialogbetrieb möglich.)

Da das System vielseitig einsetzbar sein soll, darf seine Speicherbelegung nicht zu starr ausgelegt sein:

- (8) Der Anwender muß die Möglichkeit haben, die Menge des von ihm zu verwendenden Speicherplatzes und seine Aufteilung für die verschiedenen Zwecke anzugeben.

Bei großen Mengen an benötigtem Speicherplatz kann nur ein Teil der Daten im Kernspeicher behalten werden:

- (8.1) Das System muß die zu bearbeitenden Daten auch außerhalb des Kernspeichers (auf Platte, Trommel oder Band) ablagern können.
- (8.2) Hierzu muß das System über ein Paging-System verfügen. Die Bedienung des Paging-Systems sollte automatisch erfolgen und keine Eingriffe des Anwenders erforderlich

machen.

- (8.3) Bei Rechnerausfall während eines Datentransportes des Paging-Systems darf die Datenstruktur nicht zerstört werden.

Im Interesse hoher Geschwindigkeit des Systems kann man noch folgende Forderungen an das System hinzufügen:

- (8.4) Die Daten sollten so abgespeichert werden, daß möglichst wenig Zugriffe auf das externe Speichermedium notwendig sind.
- (8.5) Es dürfen keine überflüssigen Zugriffe auf das externe Speichermedium vorkommen.

Die verlangte Vielseitigkeit des Systems macht es nötig, daß der Anwender auf zwei verschiedene Arten mit dem System in Kontakt treten kann: direkt, d.h. über Daten eines Eingabe-/Ausgabe-Mediums, und über ein Anwendungsprogramm:

- (9) Das System muß auf zwei Arten ansprechbar sein:
1. durch Unterprogramm-Aufrufe mit Parameter-Übergabe von einem Programm des Anwenders; Übergabe der Ergebnisse erfolgt wieder an dieses Programm
 2. durch Eingabe von Befehlen und Daten von einem Eingabe-Medium; Übergabe der Ergebnisse erfolgt auf ein Ausgabe-Medium .

Für die Bequemlichkeit des Anwenders sei hinzugefügt:

- (9.1) Das System muß Unterprogramme zur Verfügung stellen, die die Ein- Und Ausgabe bewerkstelligen.

Schließlich sind noch drei Forderungen zu beachten:

- (10) Das System sollte so aufgebaut sein, daß nötig werdende Änderungen mit vertretbar kleinem Arbeitsaufwand ausgeführt werden können.
- (11) Das System soll so formuliert sein, daß es auf möglichst vielen Maschinen mit möglichst wenig Änderungen läuft.

Speziell bei kleineren Maschinen ist noch wichtig:

- (12) Das System soll in einer Weise zur Verfügung gestellt werden, die es dem Anwender gestattet, nur den jeweils benötigten Teil im Kernspeicher zu halten.

2.1.2 Realisierungsweise

Ausgehend von den in 2.1.1 aufgestellten Forderungen soll nun die hier verwendete Art der Implementierung des Systems entwickelt werden.

Forderung (11), der Maschinenunabhängigkeit, kann am besten dadurch entsprochen werden, daß das System in einer der weitverbreiteten höheren Programmiersprachen formuliert wird. Da aufgrund der Forderung (2), der Adressierung über Zeichenkettenvariable, eine Textverarbeitung nötig ist, scheidet ALGØL 60 aus. CØBØL scheidet ebenfalls aus, da das System auch auf kleineren Anlagen, die meist nicht mit CØBØL-Compilern ausgerüstet sind, ohne große Änderungen lauffähig sein soll. Aus dem gleichen Grund scheidet auch PL/I aus. Aus diesen Erwägungen heraus wurde das System in weitgehend maschinenunabhängigem FØRTRAN IV programmiert. Da ein Einsatz des Systems auf dem Rechner Varian V73 mit 16 Bit Wortlänge beabsichtigt ist, das System aber gleichzeitig ohne große Änderungen auch auf Maschinen anderer Struktur und Wortlänge (16 Bit) lauffähig sein soll, wurden generell nur 15 Bit jedes Wortes ausgenutzt. (Das höchste Bit bleibt wegen seiner Bedeutung als Vorzeichenbit unbenutzt, da sonst bei bitweiser Interpretation der Maschinenworte wegen unterschiedlicher Komplementdarstellung negativer Zahlen auf verschiedenen Maschinen Schwierigkeiten auftreten könnten.) Character werden generell im Interncode der V73 dargestellt; die Umcodierung wird bei anderen Maschinen durch eigene Ein- und Ausgabe-Routinen übernommen. (Siehe auch Abschnitt 3.5.3).

Die Implementierung in FØRTRAN IV erlaubt es auch, die Forderungen (10) und (12) relativ leicht zu erfüllen: Da in FØRTRAN jedes Unterprogramm eine in sich geschlossene Programmeinheit bildet, kann das System in abgeschlossene, überschaubar kleine Teile aufgeteilt werden, die nur über Parameter und CØMMØN miteinander kommunizieren. Änderungen an einer Stelle wirken sich direkt nur auf das geänderte Unterprogramm aus, und weiterreichendere Folgen der Änderung lassen sich durch Überprüfen der Parameter und des CØMMØN relativ leicht

kontrollieren. Werden ferner die Unterprogramme in Form einer Bibliothek angelegt, so daß sich der Loader nur die jeweils benötigten Unterprogramme in den Kernspeicher zu holen braucht, so ist auch Forderung (12) erfüllt. Durch die Verwendung von ØVERLAYS läßt sich der benötigte Kernspeicher noch weiter reduzieren.

Forderung (9) kann schließlich dadurch erfüllt werden, daß zusätzlich zu den Unterprogrammen der Bibliothek, die vom Anwenderprogramm aufgerufen werden können, ein Hauptprogramm zur Verfügung gestellt wird, das Daten einlesen und als Befehle, d.h. Aufrufe der entsprechenden Unterprogramme mit geeigneten Parametern, interpretieren kann. (Siehe auch Abschnitt 3.2.1).

Zusammenfassung: Das System wird als Bibliothek von vorübersetzten FØRTRAN IV -Unterprogrammen zusammen mit einem Hauptprogramm für Dialogverkehr zur Verfügung gestellt. Die Programme sind auf jedem Rechner lauffähig, der genügend viel Kernspeicher (ca. 20 - 30 k), mindestens 16 Bit Wortlänge und duale Zahlendarstellung besitzt.

2.1.3 Bezeichnungen

In diesem Abschnitt sollen einige im Folgenden immer wieder verwendete Bezeichnungen definiert werden.

Name: = Zeichenkettenvariable, die ein Datenelement bezeichnet

Entity: = ein Datenelement, zusammen mit allen Namen, die es bezeichnen

Single-Value-Entity: = einfaches Datenelement; Entity, die weder Menge noch Relation ist

Multi-Value-Entity: = Entity, die eine Menge repräsentiert

Synonyme: = Namen, die dieselbe Entity bezeichnen

Datenspeicher: = der Speicherbereich, der die Entities enthält und direkt (also nicht über Namen) adressiert wird

ID: = Adresse einer Entity im Datenspeicher

Namenspeicher: = der Speicherbereich, der die Zuordnungstabellen der Namen und IDs enthält und über die Namen adressiert wird

Für die folgenden Definitionen muß der Begriff Relation präzisiert werden:

(Binäre) Relation: = Sei E die Menge aller Entities,

$D \subseteq E \times E$ heißt dann binäre Relation.

Diese Relation wird durch die Abbildung

$r : D \rightarrow \{r\} \times D$ mit

$(e_1, e_2) \in D \quad r(e_1, e_2) = (r, e_1, e_2)$ repräsentiert.

Entry(einer Relation): = e_1 und e_2 werden Entries der Relation r genannt

Triade: = ein Tripel $ABC=r \ e_1 \ e_2$ von Entities, das im Bildbereich der Relation r liegt. Triaden werden als Tripel von IDs im Speicher dargestellt.

Triadenspeicher: = der Speicherbereich, in dem die Triaden der abgespeicherten Relationen stehen; dieser Speicherbereich wird zuerst über einen Hash-Algorithmus und über Ring-Strukturen adressiert.

Page-Tabelle (Seitentabelle): = Daten-, Namen- und Triadenspeicher sind in Seiten unterteilt von denen sich jeweils nur je eine im Kernspeicher befindet. Der aktuelle Zustand dieser Seiten wird in der Page-Tabelle notiert.

Bezeichnungen wie Ring, Kette, Hash-Adressierung werden im Abschnitt 2.2.2 an den Stellen erläutert, an denen diese Strukturen beschrieben werden.

2.1.4 Funktionelle Gruppen von Unterprogrammen

Um überhaupt mit einer Datenstruktur arbeiten zu können, muß man in der Lage sein, Daten in die Struktur einzuschreiben, zu verändern und zu löschen. Daraus ergibt sich die Notwendigkeit für

eine erste Gruppe von Unterprogrammen:

(G1) Programme, die den Datenbestand der Struktur vergrößern, ändern oder verkleinern:

- (G1.1) STore Single - Value - Entity : STSNGL
- (G1.2) CHange Single - Value - Entity : CHSNGL
- (G1.3) DeLete Single - Value - Entity : DLSNGL

Aus den Forderungen (1) bis (4), speziell aus (4.1) ergibt sich die Notwendigkeit für eine zweite Gruppe von Unterprogrammen:

(G2) Programme, die die Verknüpfung von Zeichenkettenvariablen und den zugehörigen Daten manipulieren:

- (G2.1) CHange NAME : CHNAME
- (G2.2) DEclare SYNonym : DECSYN
- (G2.3) DeLete NAME : DLNAME

Forderung (5), speziell der erste Teil von (5.1) führt zu einer dritten Gruppe von Unterprogrammen:

- (G3) Programme, die Mengen und ihre Elemente manipulieren:
- (G3.1) DEclare SET : DECSET
 - (G3.2) STore ELEment : STELEM
 - (G3.3) DeLete ELEment : DLELEM
 - (G3.4) DeLete SET : DLSET

Forderung (6), speziell der erste Teil von (6.1) macht eine vierte Gruppe von Unterprogrammen notwendig:

- (G4) Programme, die Relationen und ihre Entries manipulieren:
- (G4.1) DEclare RELation: DECREL
 - (G4.2) STore RELation - Entry : STRELE
 - (G4.3) DeLete RELation - Entry : DLRELE

Da der Anwender auch in der Lage sein muß, die ins System gesteckte Information wieder zu erhalten, entsprechen den Gruppen (G1) bis (G4) vier weitere Gruppen von Unterprogrammen, von denen eine allerdings nur aus einem Unterprogramm besteht.

(G5) Ein Programm, das gezielt Daten aus der Struktur liest:

(G5.1) ReaD ENTity : RDENT

(G6) Programme, die gezielt Namen aus der Struktur lesen:

(G6.1) ReaD NAME : RDNAME

(G6.2) ReaD SYNonym : RDSYN

(G7) Programme, die überprüfen, ob eine Elementbeziehung vorliegt und die Elemente einer Menge auflisten :

(G7.1) FinD ELEment : FDELEM

(G7.2) LIst ELEment : LIELEM

(G8) Programme, die Fragen über Relationen beantworten und Entries auflisten:

(G8.1) FinD RELation - Entry : FDRELE

(G8.2) FinD Ring - TØP : FDRTØP

(G8.3) FinD NEXT Ring-Element : FDNEXT

Damit sind die restlichen Teile der Forderungen (4) bis (6) erfüllt. Forderung (7) beinhaltet die Notwendigkeit von Systemdienst-Programmen. Diese Programme sind so angelegt, daß sie auch Forderung (8) erfüllen :

(G9) Programme zur Systeminitialisierung, zum Datentransport und zur Speicherbereinigung (Garbage Collection) :

(G9.1) SYstem - INiTialize : SYSINT

(G9.2) PAGE - Transport : PAGE

(G9.3) SYstem - Garbage - Collection : SYSGCL

(G9.4) SYstem - Freigabe : SYSØUT

Forderung (9) ist dadurch zu erfüllen, daß zusätzlich zu den genannten Routinen ein Hauptprogramm zur Verfügung steht, das den Dialogverkehr abwickelt:

(D1) Ein Programm, das aus Daten der Eingabe Unterprogramm-Aufrufe erzeugt und Ausgabe-Daten der Unterprogramme in verständlicher Form ausgibt:

(D1.1) Program DIALØG

Schließlich wird Forderung (9.1) durch spezielle Ein- und Aus-

gabe-Prozeduren erfüllt:

(D2) Programme, die ein- und ausgehende Daten geeignet umcodieren, Fehlermeldungen und Informationen für den Anwender ausgeben und Fragen an den Anwender stellen:

- (D2.1) Data - IN - CØDe : DINCØD
- (D2.2) Data - ØUT - CoDe : DØUTCD
- (D2.3) ERRØR : ERRØR
- (D2.4) ASK : ASK

(D3) Programme, die umcodierte Daten zu Parametern für Unterprogramm-Aufrufe aufbereiten; diese Routinen werden ausschließlich vom Programm DIALØG aus aufgerufen und können als Teile dieses Programmes verstanden werden:

- (D3.1) ALPHA Namen - Parameter : ALPHAN
- (D3.2) ALPHA Daten - Parameter : ALPHAD
- (D3.3) NUMerische Parameter : NUM

Die Zurückführung eines großen Teiles dieser Programme auf relativ wenige Grunddienste und die Verwendung einiger Hilfsroutinen wird im übernächsten Abschnitt behandelt, nachdem im nächsten Abschnitt die Struktur der benutzten Speicherbereiche und der Aufbau des Paging-Systems beschrieben wurde.

2.2 Speicherstruktur

2.2.1 Das Paging-System

Das DATAS-System arbeitet auf drei großen, in Seiten festgelegter Länge eingeteilten Speicherbereichen:

1. dem Namenspeicher, der die Zuordnung von Namen und IDs enthält und über die Namen adressiert wird;
2. dem Triadenspeicher, der die Information über Relationen enthält; dieser Speicherbereich wird zuerst über einen Hash-Algorithmus und intern über Ring-Strukturen adressiert;
3. dem Datenspeicher, der die Daten und Namen der Entities enthält und über die IDs adressiert wird.

Der Umfang der zu speichernden Daten macht in den meisten Fällen eine Einteilung dieser drei Speicherbereiche in Seiten nötig, von denen nur die gerade benötigte im Kernspeicher steht, während alle Seiten auf Platte stehen. Dieser Sachverhalt macht die Existenz eines Paging- Systems nötig, das den Forderungen (8.1) bis (8.5) genügen muß. Das hier verwendete Paging-System besteht im wesentlichen aus einer Tabelle, die den aktuellen Zustand der einzelnen Seiten beschreibt und einer Routine, die, falls nötig, eine der im Kernspeicher vorhandenen Seiten auf ihren Platz auf der Platte hinausschreibt und die im Augenblick benötigte Seite von der Platte in den Kernspeicher liest (Page-Routine). Die nötigen Abfragen über das Vorhandensein freier Seiten und, im Datenspeicher, über den freien Raum auf den einzelnen Seiten, werden von den Routinen, die das Paging-System in Anspruch nehmen, selbst ausgeführt. Die Page-Tabelle ist in diesen Routinen als labeled COMMON vereinbart und kann somit die benötigten Informationen liefern. Lediglich die Abfrage, ob die benötigte Seite schon im Kernspeicher steht, wird von der Page-Routine ausgeführt, um überflüssige Plattenzugriffe mit Sicherheit zu vermeiden.

2.2.1.1 Die Seitentabelle (Page-Tabelle)

Da der Speicherplatz für die Seitentabelle in FØRTRAN IV statisch zur Verfügung gestellt werden muß, kann aus Gründen der Speicherersparnis die Seitentabelle nicht beliebig lang gemacht werden. In der vorliegenden DATAS-Implementierung wurde zunächst einmal Platz für 100 Seiten vorgesehen; die Seitentabelle wurde, da sie noch 9 Worte für andere Zwecke benötigt, als INTEGER PAGTAB(109) vereinbart. In der Systeminitialisierungs-Routine SYSINT wurde eine Sicherheitsabfrage eingebaut, so daß maximal nur 100 Seiten als Speichergröße vereinbart werden können. (Es ist selbstverständlich möglich, ohne Änderung am Programm-System weniger Speicher zu vereinbaren.) Durch Abändern der Sicherheitsabfrage in SYSINT und der Vereinbarung INTEGER PAGTAB(109) in den Routinen, die

Zugriff auf PAGTAB haben, ist es möglich, die Seitenanzahl auf mehr als 100 Seiten zu vergrößern. Weitere Änderungen sind hierzu nicht nötig, da nirgends Bezug auf die Länge des Feldes PAGTAB genommen wird. Die Seitentabelle ist folgendermaßen aufgebaut:

PAGTAB (1) enthält die Nummer der aktuellen, d.h. gerade im Kernspeicher befindlichen Seite des Namenspeichers
 PAGTAB (2) enthält die Nummer der aktuellen Seite des Triadenspeichers
 PAGTAB (3) enthält die Nummer der aktuellen Seite des Datenspeichers
 PAGTAB (4) ist zur Zeit noch unbenutzt
 PAGTAB (5) enthält die Nummer der nächsten freien Seite des Triadenspeichers
 PAGTAB (6) enthält die Nummer der nächsten freien Seite des Datenspeichers
 PAGTAB (7)=Anzahl der Seiten des Namenspeichers
 PAGTAB (8)=PAGTAB (7) + Anzahl der Seiten des Triadenspeichers
 PAGTAB (9)=Gesamtzahl der zur Verfügung stehenden Seiten
 PAGTAB (I), $10 \leq I \leq \text{PAGTAB (7)} + 9$ ist zur Zeit noch unbenutzt
 PAGTAB (J), $\text{PAGTAB (7)} + 10 \leq J \leq \text{PAGTAB (8)} + 9$ beschreibt den Zustand der Seite :

- 0 : die Seite ist besetzt und enthält keine gelöschten Triaden
- 1 : die Seite ist besetzt und enthält gelöschte Triaden
- 2 : die Seite ist frei und enthält keine gelöschten Triaden
- 3 : die Seite ist frei und enthält gelöschte Triaden

PAGTAB(K), $\text{PAGTAB (8)} + 10 \leq K \leq \text{PAGTAB (9)} + 9$ enthält die Adresse des ersten freien Wortes der Seite, bzw. ist 0, wenn die Seite vollgeschrieben ist.

Falls keine freie Seite in einem der drei Speicherbereiche mehr vorhanden ist, so wird das zugehörige Wort PAGTAB(M), $M = 5$ bzw. 6 auf 0 gesetzt. Die Zellen PAGTAB(4) und PAGTAB(I) sind für den weiteren Ausbau des Systems vorgesehen. (Siehe auch Abschnitt 2.2.2.1)

2.2.1.2 Der Seitentransport (Page-Routine)

Die einzelnen Seiten von Namen-, Triaden- und Datenspeicher können durch Aufruf der Routine PAGE beliebig aus dem Kernspeicher auf ihren Platz auf dem externen Speichermedium hinausgeschrieben werden oder von ihm in den Kernspeicher gelesen werden. Dabei führt die Page-Routine nur dann einen Seitentransport durch, wenn die benötigte Seite von der aktuellen, d.h. gerade im Kernspeicher befindlichen Seite verschieden ist, da dieser Plattenzugriff der um Größenordnungen langsamste Befehl des Systems ist. Die Routine PAGE wird im Abschnitt 3.3.9.2 ausführlich beschrieben.

2.2.2 Die Strukturen der einzelnen Seiten

2.2.2.1 Namenspeicher

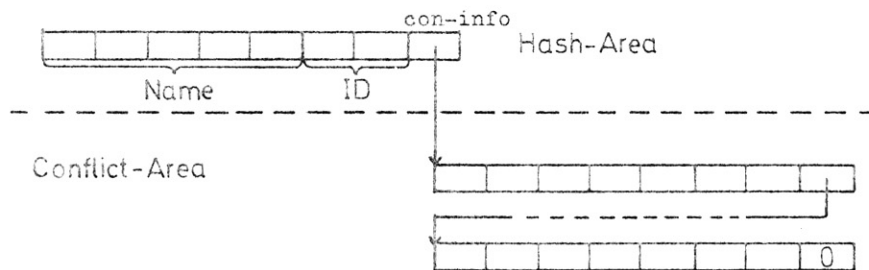
Der Namenspeicher ist als labeled COMMON vereinbart, und zwar als INTEGER IN(2048).

Der erste Teil dieser Seite ist hash-adressiert, die aus den Characters eines Namens gebildete Hash-Funktion wird in Abschnitt 3.5.1.2 beschrieben. Dieser Teil des Namensspeichers ist nicht vorstrukturiert. Jeder Eintrag wird (bis auf Conflict-Fälle) an der von der Hash-Funktion gelieferten Adresse gemacht. Diese Adresse besteht aus zwei Zahlen:

IPAGE, der Nummer der Seite des Namenspeichers
IADR, der Adresse des ersten zu besitzenden Wortes
auf der Seite

Da jeder Name aus maximal 10 Characters besteht, werden für die Abspeicherung des Namens 5 Worte benötigt. Zwei weitere Worte nehmen die beiden Zahlen des ID der zugehörigen Entity auf. Ein achttes Wort schließlich enthält Conflict-Information. Darunter ist folgendes zu verstehen: Da Hash-Funktionen nicht

injektiv sind, kann es vorkommen, daß ein Name als Hash-Adresse die Adresse einer schon besetzten Zelle zugewiesen bekommt. In diesem Falle wird der Name im zweiten Teil der Seite untergebracht, in der sogenannten Conflict-Area. Das achte Wort der entsprechenden Zelle in der Hash-Area, das normalerweise 0 ist, enthält jetzt die Adresse der ersten ihm zugehörigen Conflict-Zelle; das achte Wort dieser ist 0, wenn es keine weiteren Conflict-Zellen zu derselben Hash-Adresse gibt; andernfalls enthält es die Adresse der nächsten zugehörigen Conflict-Zelle. Eine solche Struktur wird hier als Conflict-Kette bezeichnet.



Die freien Zellen der Conflict-Area bilden ebenfalls eine Kette, die sogenannte vacant- Kette, deren Ketten-Information ebenfalls im achten Wort jeder Zelle steht; eine 0 hier zeigt ebenfalls das Ende der Kette an. In IN(2047) steht die Adresse des Anfangs der Conflict-Area, in IN(2048) die Adresse des Anfangs der vacant-Kette. Ist also IN(2048) gleich 0, so ist die Conflict-Area besetzt; wird dennoch versucht, einen Namen in die Conflict-Area zu schreiben, so erfolgt die Fehler-

meldung "NAMENSPEICHER VÖLL" und die Abspeicherung der entsprechenden Entity bzw. des Synonyms unterbleibt. Es ist geplant, für diesen Fall einen Overflow-Bereich von einigen Seiten anzulegen, die vorliegende Implementierung enthält diesen Komfort jedoch noch nicht, wenn auch die Seitentabelle schon so ausgelegt ist, daß der Anschluß eines Overflow-Bereiches möglich ist.

Der Anfang der Conflict-Area wird von der System-Routine SYSINT für alle Seiten des Namensspeichers auf den gleichen Wert gesetzt; er kann jedoch auch vom Anwender durch explizites Setzen der Variablen IN(2047) und Aufrufen der Routine NSINIT für jede Seite getrennt festgesetzt werden.

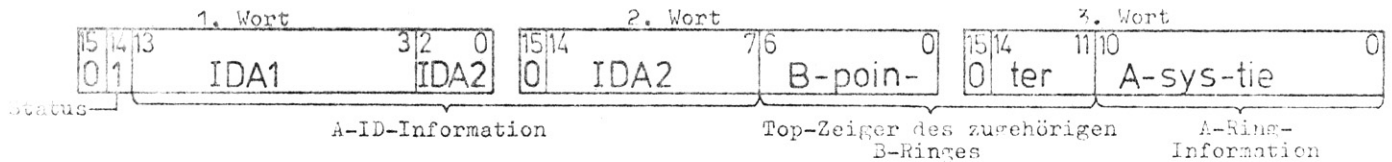
Beim Löschen einer Zelle wird die Namen- und ID-Information 0 gesetzt, in der Hash-Area ebenfalls die Conflict-Information, in der Conflict-Area wird die Zelle dagegen in die vacant-Kette eingefügt. Falls die Zelle Glied einer Conflict-Kette war, so wird sie vor dem Löschen aus dieser Kette herausgelöst und die Kette upgedatet. Eine Garbage-Collection ist daher für den Namenspeicher nicht nötig.

2.2.2.2 Triadenspeicher

Die Struktur des Triadenspeichers entspricht weitgehend der in / 11 / beschriebenen; Unterschiede werden in Abschnitt 5 genauer erläutert.

Der Triadenspeicher ist als blank COMMON vereinbart, und zwar als INTEGER IP (2048). Der erste Teil jeder Seite, der sogenannte freie Speicher F1 / 6 /, ist als doppelte Ringstruktur ausgebildet; der zweite Teil ist hash-codiert und wird Hash-Area HAS genannt. Der dritte Teil schließlich ist die zugehörige Conflict-Area; er wird als freier Speicher F2 bezeichnet und kann mehrere Ringstrukturen umfassen.

Im freien Speicher F1 werden die IDs der abgespeicherten Relation-Entities aufbewahrt; behält man die in Abschnitt 2.1.3 eingeführte Bezeichnung ABC für Triaden bei, so befinden sich also die A-IDs in F1. Sie sind in Form eines sogenannten occupied-Ringes gespeichert. Dabei ist unter einem Ring folgende Struktur zu verstehen: Eine Kette wird um ein weiteres Element erweitert, das jedoch nicht als zur Kette gehörig betrachtet wird. Dieses Element wird als Kopf des Ringes bezeichnet; es steht in gewisser Analogie zum Start-Register IN(2048) der vacant-Kette des Namensspeichers. Im Kopf des Ringes steht nun die Adresse der ersten Zelle der Kette, die als Top des Ringes bezeichnet wird; man sagt auch, der Kopf enthält einen Zeiger (pointer) auf das Ring-Top. Das Ring-Top enthält nun die normale Information über die Adresse der nächsten Zelle der Kette; dieser Zeiger wird auch als sys-tie bezeichnet. Ferner wird das Ring-Top noch durch Setzen eines besonderen Bits, des Top-Bits, in dem Bereich der Zelle, der die Ring-Information enthält, als Top gekennzeichnet; bei allen Nicht-Top-Zellen des Ringes ist dieses Bit 0. Im Gegensatz zur Kette ist nun das sys-tie der letzten Zelle der Kette, das Bottom des Ringes, nicht 0, sondern es enthält einen Zeiger auf den Kopf des Ringes (nicht auf das Top!); damit ist die Kette zum Ring geschlossen. Wie das Top wird auch das Bottom des Ringes durch Setzen eines besonderen Bits, des Bottom-Bits, gekennzeichnet. Man hat also folgende Struktur:



Das (Vorzeichen-)Bit 15 bleibt aus den in Abschnitt 2.1.2 erörterten Gründen unbesetzt; Bit 14 gibt den Status der Zelle an:

Status-Bit = 0 : Zelle ist leer

Status-Bit = 1 : Zelle ist besetzt (enthält Information oder ist gelöscht)

Auf die Verwendung von Top- und Bottom-Bits konnte hier, wie auch in den vacant-Ringen, verzichtet werden.

Alle zu derselben A-Zelle gehörenden B-Zellen sind ebenfalls in Form eines Ringes organisiert, dessen Kopf die zugehörige A-Zelle ist. Die Adressen der B-Zellen werden mittels der in Abschnitt 3.5.3.2 besprochenen Funktion IHASH aus den IDs von A und B berechnet. Diese B-Zellen liegen (soweit keine Conflict-Situationen auftreten) in der Hash-Area HAS, die, wie auch die entsprechende Area im Namenspeicher, nicht vorstrukturiert ist.

Der freie Speicher F2 enthält dagegen wieder einen vacant-Ring, dessen Kopf-Zelle die erste Zelle von F2 ist. Hier sind zunächst einmal alle die B-Zellen abgespeichert, die aufgrund von Conflict-Situationen nicht in der Hash-Area abgespeichert werden konnten. Alle zu einer Hash-Adresse gehörenden B-Zellen sind in Form eines sogenannten Conflict-Ringes abgespeichert, dessen Kopf in HAS steht, während alle anderen Zellen in F2 stehen. Jede B-Zelle eines Conflict-Ringes gehört somit zwei unabhängi-

gen Ringen an: dem B-Ring, dessen Kopf die zugehörige A-Zelle ist, und dessen Ring-Information weiterhin als sys-tie bezeichnet wird, und dem Conflict-Ring, dessen Kopf eine B-Zelle in HAS ist und dessen Ring-Information als con-tie bezeichnet wird, um Verwechslungen mit dem sys-tie zu vermeiden.

Die C-Zellen der Triaden können auf zwei verschiedene Arten abgespeichert werden:

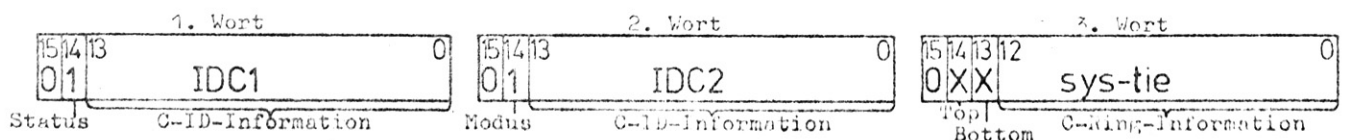
1. Wenn zu einem Paar AB nur ein einziger C-Entry gehört, so wird sein ID direkt in der B-Zelle abgespeichert. Die B-Zelle wird dann als Terminal bezeichnet.
2. Gehören mehrere C-Entries zu demselben Paar AB, so werden ihre IDs in Form eines Ringes, des sogenannten C-Ringes, abgespeichert, der in F2 liegt und die zugehörige B-Zelle als Kopf hat.

In F2 kommen somit zwei verschiedene Typen von Zellen vor: B-Zellen mit einer Länge von 6 Worten und C-Zellen mit einer Länge von drei Worten. Zwischen diesen beiden Zell-Typen wird durch das Bit 14 des zweiten Wortes, das sogenannte Modus-Bit unterschieden.

Modus-Bit = 0 : B-Zelle, 6 Worte lang

Modus-Bit = 1 : C-Zelle, 3 Worte lang

Eine C-Zelle hat somit folgenden Aufbau:

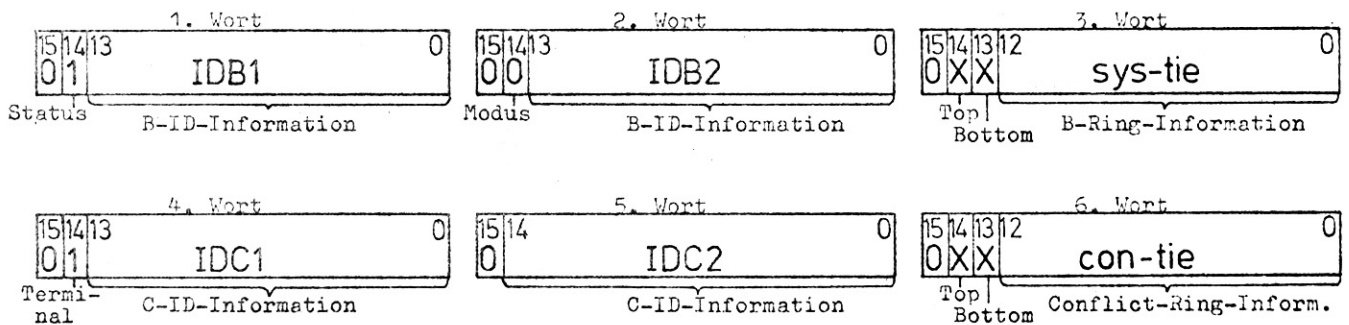


B-Zellen sind dagegen verschieden aufgebaut, je nachdem, ob es sich um ein Terminal handelt oder nicht. Der Unterschied wird in Bit 14 des 4. Wortes, dem sogenannten Terminal-Bit, markiert.

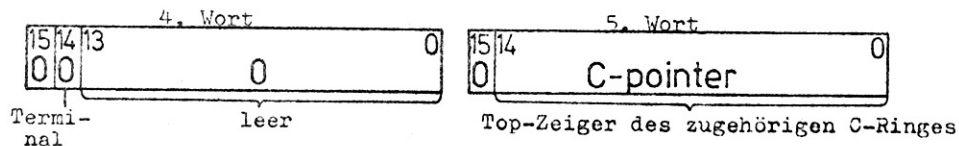
Terminal-Bit = 0 : Kopf eines C-Ringes

Terminal-Bit = 1 : Terminal

Ein Terminal besitzt somit folgenden Aufbau:



Eine Nicht-Terminal-Zelle unterscheidet sich hiervon nur im Aufbau des 4. und 5. Wortes:

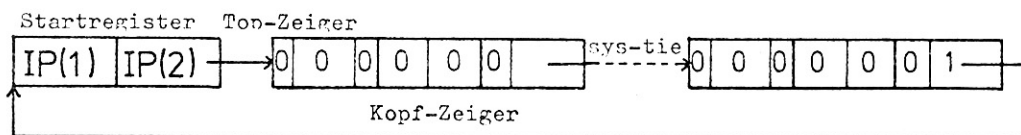


Top- und Bottom-Bit des 6. Wortes sind bei allen B-Zellen in HAS gleich 0. Existiert zu einer dieser Zellen ein Conflict-Ring, so enthält con-tie den Top-Zeiger dieses Ringes; andernfalls ist con-tie ebenfalls gleich 0.

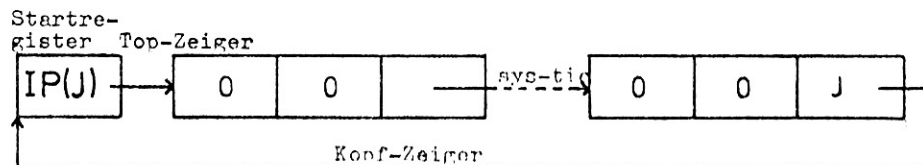
Alle Zeiger geben die Adressen der angegebenen Zellen direkt an, im Gegensatz zu / 6 /, wo drei Adressbereiche, nämlich F1, HAS und F2 unterschieden werden und wo durch ein eigenes Area-Bit unterschieden wird, ob sich eine B-Adresse auf HAS oder F2 bezieht. Die hier verwendete direkte Adressierung hat demgegenüber den Vorteil, mit geringerem Rechenaufwand Adressen bestimmen zu können und alle Abfragen, auf welchen Teilbereich sich eine gegebene Adresse bezieht, überflüssig zu machen. Die Grenzen der einzelnen Teilbereiche werden wieder, analog zum Namenspeicher, auf der Seite selbst festgehalten, und zwar enthält IP(2047) die Adresse des letzten Wortes von F1, IP(2048) die Adresse des letzten Wortes von HAS. Diese Grenzen werden von der System-Initialisierungs-Routine SYSINT für alle Seiten gleich festgesetzt; sie können aber auch vom Anwender durch explizites Setzen der Variablen IP(2047) und IP(2048) und anschließendes Anrufen von F1INIT, HSINIT und F2INIT für jede Seite getrennt festgesetzt werden.

Anschließend seien noch einmal die einzelnen Teilstrukturen des Triadenspeichers dargestellt:

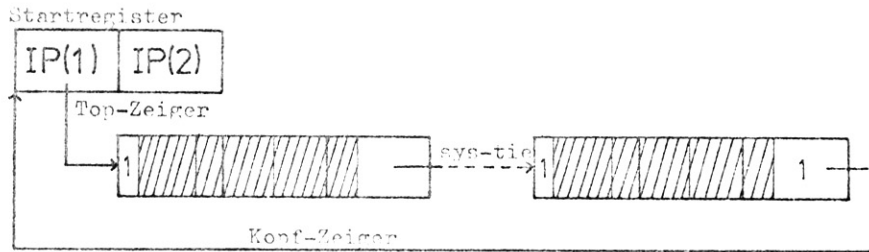
F1, vacant-Ring



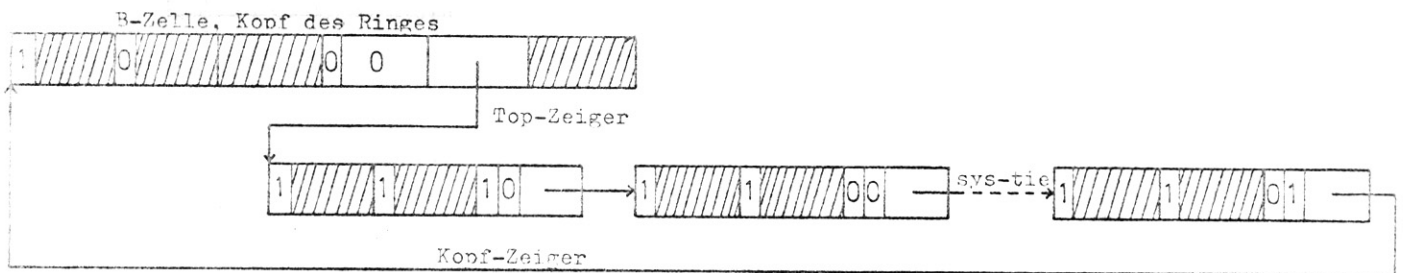
F2, vacant-Ring ($J = IP(2048)+1$)



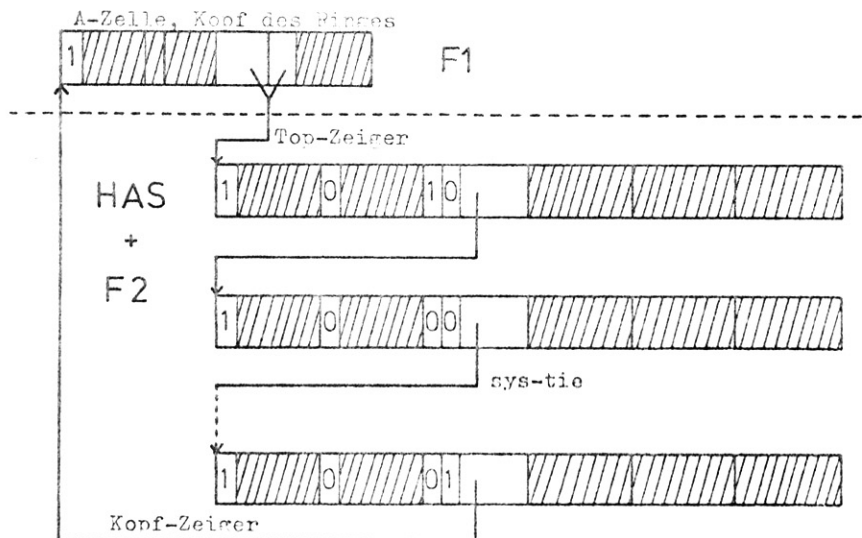
F1, occupied-Ring (A-Ring):



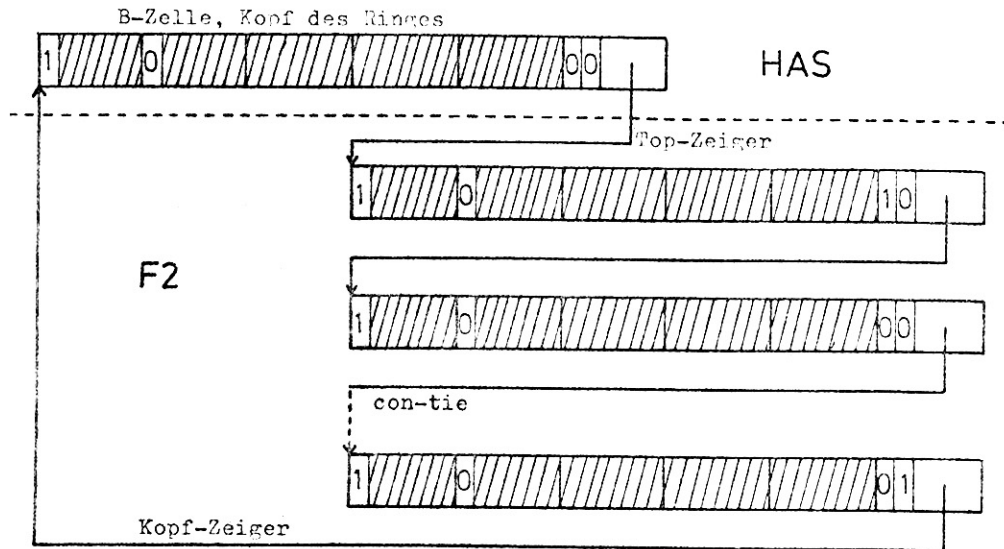
C-Ring (Kopf in HAS oder F2, Ring in F2):



B-Ring (in HAS und/oder F2, Kopf in F1):



Conflict-Ring (Kopf in HAS, Ring in F2):



Teile der Zellen, die für die betreffende Ringstruktur unwichtig sind, wurden schraffiert. Das Löschen einer Zelle besteht im Nullsetzen ihrer ID-Information, bei Terminals auch der C-ID-Information. Da das Status-Bit nicht mitgelöscht wird, sind gelöschte, aber noch nicht freigegebene Zellen dadurch kenntlich, daß das 1. Wort den Wert 2^{14} hat. Dies gilt gleichermaßen für A-, B- und C-Zellen. Die Freigabe der Zellen, also das Herauslösen aus den Ringstrukturen und (in F1 bzw. F2) das Eingliedern in den vacant-Ring bzw. das völlige Nullsetzen der Zelle (in HAS) wird von der Garbage-Collection-Routine GARCØL für eine Seite gemacht. Die Speicherbereinigung des gesamten Triadenspeichers wird von der System-Routine SYSGCL vorgenommen. Diese Routinen werden in den Abschnitten 3.4.2.4 und 3.3.9.3 besprochen.

2.2.2.3 Datenspeicher

Der Datenspeicher ist ebenfalls in Seiten der Länge 2k Worte eingeteilt. Die im Kernspeicher befindliche Seite ist als labeled ~~COMMON~~ vereinbart, und zwar als INTEGER IE(2048). Die ersten 257 Worte von IE bilden, analog zu F1, eine doppelte Ringstruktur, die aus einem occupied- und einem vacant-Ring besteht. IE(1) ist Startregister beider Ringe; die Bits 13 bis 7 enthalten den Top-Zeiger des occupied-Ringes, Bits 6 bis 0 den des vacant-Ringes. Die Zellen des occupied-Ringes bestehen, wie auch die des vacant-Ringes, aus je zwei Worten. Das erste Wort einer vacant-Zelle ist 0, das einer occupied-Zelle enthält in den Bits 12 bis 0 die Adresse des ersten Wortes des zugehörigen Datenblocks, der irgendwo hinter IE(258) abgespeichert ist. Die Bits 14 und 13 des ersten Wortes enthalten die Klasse, der die Entity angehört, und zwar bedeuten:

Klasse-Bits = 1 : Relation-Entity
 Klasse-Bits = 2 : Single-Value-Entity
 Klasse-Bits = 3 : Multi-Value-Entity

Das zweite Wort jeder Zelle enthält das sys-tie des entsprechenden Ringes und ist 0 bei den Bottom-Zellen. Da IE nun gerade 128 solcher Zellen besitzt und da das erste Wort jeder Zelle eine gerade Adresse besitzt, genügen zur Adressierung jeder Zelle genau 7 Bit. Das zweite Wort jedes ID enthält genau diese 7 Bit, ebenso wie auch alle Ring-Zeiger der beiden Ringe. Das erste Wort jedes ID enthält die Seitennummer der Entity im Datenspeicher. Da für jedes Wort des ID aufgrund des Aufbaues der A-Zellen des Triadenspeichers maximal 11 Bit zur Verfügung stehen, kann das System zur Zeit maximal 2^{18} Entities bearbeiten. Bei anderer Einteilung der A-Zellen ist jedoch aufgrund des soeben beschriebenen Aufbaues der Ringstrukturen des Datenspeichers eine Erweiterung auf maximal 2^{22} Entities möglich, was aber schon einen Plattenspeicher von ca. $2^{27} = 128\text{M}$ Worten nötig macht. Daher wurde bisher auf

diese Ausbaustufe verzichtet.

Die Worte auf der Seite, die Indizes ≥ 259 haben, dienen zur Aufnahme der Daten und Namen der abgespeicherten Entities. Die Daten sind fortlaufend abgespeichert, und zwar in Blöcken variabler Länge, von denen jeder eine Entity beschreibt und somit durch eine Zelle im ersten Teil der Seite adressiert wird. Die Blöcke haben für Entities der verschiedenen Klassen einen etwas unterschiedlichen Aufbau; ihre Struktur ist jedoch für alle Entities gleich: Das 1. Wort jedes Blockes enthält einen Zeiger auf den Beginn des nächsten Blocks, das erste Wort des letzten Blocks zeigt auf den Anfang des freien Datenspeichers. In den Bits 14...11 des zweiten Wortes steht die Anzahl der Synonyme dieser Entity, also die Anzahl der Namen, die diese Entity adressieren. Die letzten Worte des Datenblocks bestehen aus Gruppen von je 5 Worten (wenigstens einer Gruppe), die diese Namen enthalten. Zwischen diesen Teilen des Datenblocks, die für alle Entities gleich sind, stehen die Daten, die die Entity enthält:

Bei Relation-Entities steht in den Bits 1 und 0 des zweiten Wortes des Datenblocks der Permutationscode der Relation, und zwar bedeutet:

Permutations-Bits = 0 : ABC	} werden abgespeichert
Permutations-Bits = 1 : ABC BCA	
Permutations-Bits = 2 : ABC BCA CAB	

Dieser Permutationscode kann für jede Relation frei gewählt werden; er wird dann aber für alle Entries der Relation gleich gesetzt. In den Bits 10 bis 2 des zweiten Wortes des Datenblocks steht schließlich die Anzahl k der Seiten im Triadenspeicher, auf denen die Relation abgespeichert ist. Die folgenden k Worte enthalten die Nummern dieser Seiten; es kann vorkommen, daß verschiedene Permutationen eines Entry auf verschiedenen Seiten im Triadenspeicher stehen; in diesem Fall

gibt es aber eine Seite, auf der alle Permutationen stehen, und nur deren Nummer wird in der Liste eingetragen. Alle folgenden Worte bis zum Beginn des Bereichs, in dem die Namen der Relation abgespeichert sind, können beliebige Daten enthalten.

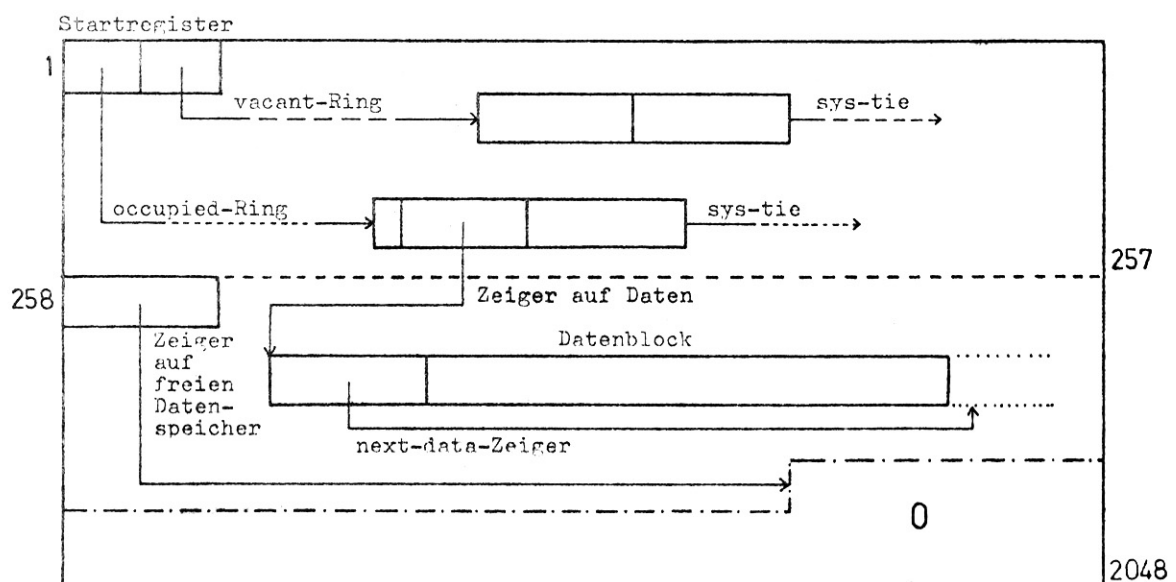
Bei Single-Value-Entities sind die Bits 10 bis 0 des zweiten Wortes des Datenblocks 0; alle folgenden Worte bis zum Beginn des Bereichs, in dem die Namen der Entity stehen, können beliebige Daten enthalten.

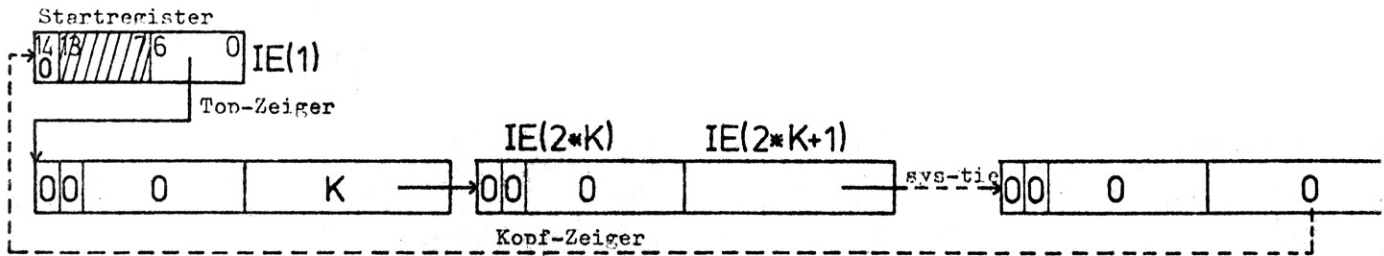
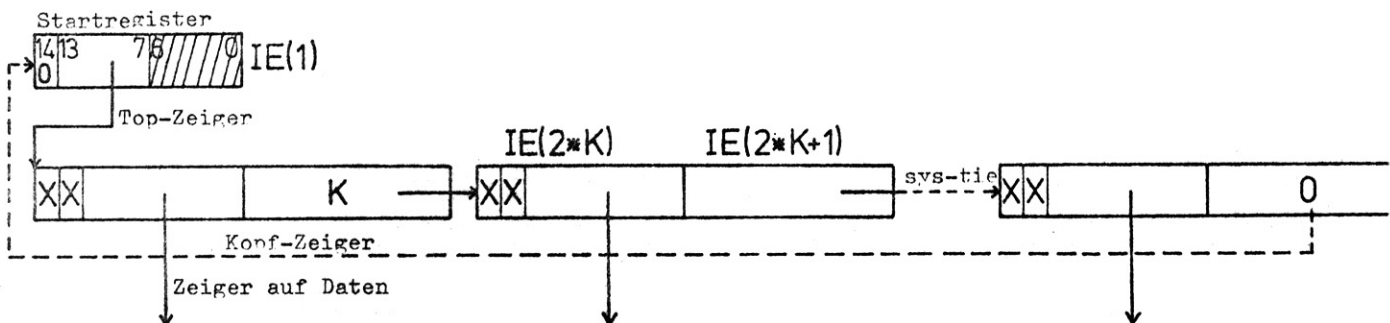
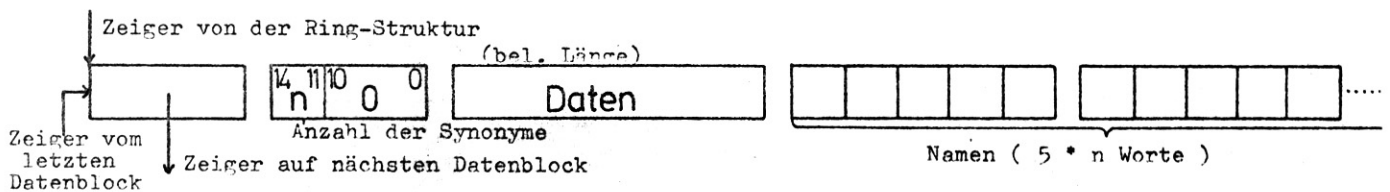
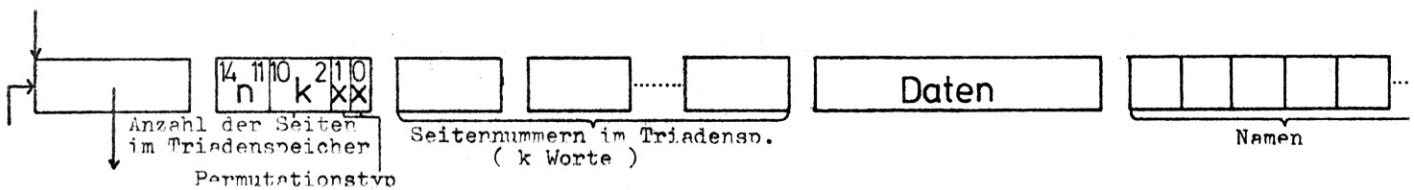
Bei Multi-Value-Entities, also Mengen, enthalten die Bits 10 bis 0 des zweiten Wortes des Datenblocks die Anzahl m der Elemente der Menge; die $2m$ nächsten Worte enthalten die IDs der Elemente der Menge. Alle weiteren Worte bis zum Beginn des Namen-Bereiches der Entity können beliebige Daten enthalten.

Das 258. Wort der Seite enthält schließlich die Adresse des ersten Wortes des freien Datenspeichers der Seite.

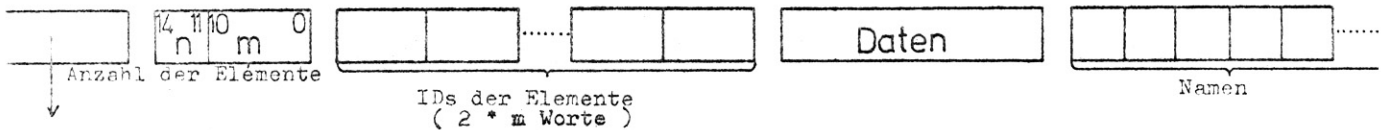
Analog zu den Ringstrukturen des Triadenspeichers seien die Strukturen des Datenspeichers dargestellt:

Einteilung der Seite:



vacant-Ring:**occupied-Ring:****Datenblock einer Single-Value-Entity:****Datenblock einer Relation-Entity:**

Datenblock einer Multi-Value-Entity:



Wird eine Entity gelöscht, so wird ihre Kopfzelle aus dem occupied-Ring herausgelöst und in den vacant-Ring eingefügt; dabei wird das erste Wort der Kopfzelle gelöscht. Die Datenblöcke aller folgenden Entities werden um die Länge der gelöschten Entity nach vorne geschoben, und die zugehörigen Datenzeiger werden entsprechend abgeändert. Das hier geschilderte Verfahren erlaubt eine lückenlose Abspeicherung der Daten auch nach dem Löschen von Entities, ohne daß sich die IDs der vom Löschvorgang nicht betroffenen Entities ändern. Ferner erlaubt es, die Datenblöcke neu abzuspeichernder Entities einfach hinten an die bereits vorhandenen Datenblöcke anzuhängen und die Datenblöcke vorhandener Entities nachträglich zu verlängern und zu verkürzen. Ferner entfällt die Notwendigkeit für eine Garbage Collection im Entity-Speicher. Nach den bisherigen Erfahrungen mit dem DATAS-System sind auch die Rechenzeiten für solche Manipulationen ausreichend kurz (≤ 100 ms auf der TR 440).

2.3 Die verschiedenen Ebenen des Systems

2.3.1 Systemdienste und Hilfsroutinen

2.3.1.1 Aufteilung in Anwender-bezogene Gruppen (Ebenen) von Unterprogrammen

Aus dem Vorangegangenen läßt sich erkennen, daß sich die Vielfalt der in 2.1.4 beschriebenen Unterprogramme zu einem großen Teil auf bestimmte Einzeltätigkeiten zurückführen

läßt; weiterhin kann man sich überlegen, daß auch diese Systemdienste zum Teil noch einfachere Tätigkeiten ausführen müssen, die auch von anderen Systemdiensten benötigt werden und daher zweckmäßigerweise in Form von Unterprogrammen zur Verfügung gestellt werden. Das soll an einem Beispiel kurz erläutert werden: Es soll mit CALL STRELE (STore RElation ENtry, (G4.2) in Abschnitt 2.1.4) ein Relationsentry abgespeichert werden. Dazu sind im Einzelnen folgende Nebentätigkeiten als Unterprogramme nötig:

1. Falls eine der Entities noch nicht existiert, muß sie erzeugt werden:
CALL STENT (STore ENTity)
Es wird somit eine Routine benötigt, die Entities abspeichert.
2. Falls die Relation schon existiert, aber der Entry auf einer neuen Seite abgespeichert werden muß, muß die Relation-Entity einen neuen Seiteneintrag erhalten. Dazu wird eine Routine benötigt, die die Daten im Entityspeicher verschiebt:
CALL DSHIFT (Data SHIFT)
3. Die Triade muß im Triadenspeicher abgespeichert werden:
CALL STØRE (STØRE Triade)
4. Dazu müssen die IDs der drei Entities bekannt sein. Es wird also eine Routine benötigt, die aus dem Namen einer Entity ihr ID berechnet:
CALL RTRNID (ReTuRN ID)
5. Falls nötig, müssen im Triadenspeicher Seiten ausgetauscht werden:
CALL PAGE
6. Falls Fehler auftreten, muß eine Fehlermeldung ausgegeben werden; es wird somit eine Routine benötigt, die notfalls Fehlermeldungen ausgibt, eventuell zusätzlich Dumps erzeugt und bei schweren Fehlern den Job abbricht:
CALL ERRØR

Für jedes der in 2.1.4 genannten Unterprogramme lassen sich so die benötigten Systemdienste aufstellen; weiterhin lassen sich aber auch für diese Systemdienste wieder die benötigten Hilfsroutinen aufstellen, so daß man für die Systemdienste und Hilfsroutinen eine sehr komplexe Verweisstruktur erhält. Den Anwender des DATAS-Systems braucht diese interne Programmstruktur des Systems nicht zu interessieren, so daß sich aus seiner Sicht nur 3 Ebenen von Programmen ergeben:

- Ebene 0 : Das Anwendungsprogramm
(Dies kann entweder das Dialog-Programm des DATAS-Systems oder ein vom Anwender geschriebenes Hauptprogramm sein.)
- Ebene 1 : Unterprogramme des Systems
(Hier sind die unter 2.1.4 aufgelisteten Programme gemeint; sie sind die einzigen Unterprogramme, die vom Anwender zu "sehen" sind.)
- Ebene 2 : Systemdienste und Hilfsprogramme
(Darunter sind alle Unterprogramme zu verstehen, die vom Anwender nicht zu "sehen" sind. Vom Dialog-Programm aus kann der Anwender diese Unterprogramme überhaupt nicht aufrufen, von einem eigenen Programm aus nur unter genauer Kenntnis der einzelnen Speicherstrukturen.)

Nach dieser Aufgliederung kann nun genauer zwischen Systemdiensten und Hilfsroutinen unterschieden werden: Unter Systemdiensten sind alle die Unterprogramme zu verstehen, die von Ebene 1 aus direkt aufgerufen werden. Alle anderen Unterprogramme, die also ihrerseits nur von Systemdiensten direkt oder indirekt aufgerufen werden, sollen im Folgenden als Hilfsprogramme bezeichnet werden. Dabei ist strikt zu beachten, daß diese vom Anwender her gesehene Einteilung der Unterprogramme

nichts mit der im Abschnitt 2.3.2 beschriebenen Verweisstruktur des Systems zu tun hat.

2.3.1.2 Systemdienste

Die Systemdienste lassen sich ebenso wie die Unterprogramme der Ebene 1 in funktionelle Gruppen aufteilen:

(S1) Programme zur Manipulation von Entities:

- (S1.1) Store ENTity : STENT
- (S1.2) DElete ENTity : DELENT
- (S1.3) Data SHIFT : DSHIFT
- (S1.4) ReTuRN ID : RTRNID

(S2) Programme zur Manipulation von Triaden:

- (S2.1) StORE Triade : STØRE
- (S2.2) DELETE Triade : DELETE
- (S2.3) RETRIeVe Triade : RETRIV
- (S2.4) GARbage COLlection : GARCØL

(S3) Programme zur Initialisierung der einzelnen Speicher:

- (S3.1) Namen Speicher INITialisieren : NSINIT
- (S3.2) F1 INITialisieren : F1INIT
- (S3.3) Hash Speicherbereich INITialisieren : HSINIT
- (S3.4) F2 INITialisieren : F2INIT
- (S3.5) Daten Speicher INITialisieren : DSINIT

(S4) Programme, die mit dem Anwender verkehren:

- (S4.1) ASK : ASK
- (S4.2) ERRØR : ERRØR

Diese beiden Programme gehören zu der Gruppe (D2) von I/O-Programmen. Man könnte außerdem noch nach der Verweisstruktur des Systems das Unterprogramm PAGE (G9.2) zu den Systemdiensten rechnen; da jedoch der Anwender die Möglichkeit haben muß, die Seiten der einzelnen Speicherbereiche selbst auszutauschen, wird diese Routine der Ebene 1 zugeordnet.

2.3.1.3 Hilfsroutinen

Wegen der starken Abhängigkeiten der einzelnen Hilfsroutinen voneinander lassen sie sich nur schwer in kleinere funktionelle Gruppen aufteilen. Daher wird im Folgenden nur eine grobe Einteilung nach den Speicherbereichen vorgenommen, auf die die Routinen arbeiten:

(H1) Programme, die mit dem Entity- und Namensspeicher arbeiten:

(H1.1) AlloCate NAME : ALCNAM

(H1.2) HASH NaMe : HASHNM

(H1.3) EXtract DATA : EXDATA

(H2) Programme, die mit dem Triadenspeicher arbeiten:

(H2.1) KEEP FRee : KEEPFR

(H2.2) A - ALLØCate : AALLØC

(H2.3) REQuest Class 1 : REQCL1

(H2.4) PUT Ring Into Entity : PUTRIE

Diese Routine könnte ebenso gut zur Gruppe (H1) gerechnet werden.

(H2.5) B-Ring-STaRT : BRSTRT

(H2.6) BacK-TIE : BKTIE

(H2.7) F2-RELease : F2REL

(H2.8) B-RETRIEval : BRETRI

(H2.9) C-RETRIEval : CRETRI

(H2.10) A-LØØK : ALØØK

(H2.11) GØ to BØTtoM : GOBØTM

(H3) Programme ohne Speicherbezug:

(H3.1) PERmutation - ØRDer-of-Request : PEØRDR

(H3.2) IHASH : IHASH

Dieses Unterprogramm ist als einziges in Form einer FUNCTION geschrieben.

(H3.3) Data-IN-CØDe : DINCØD

(H3.4) Data-ØUT-CoDe : DØUTCD

Die beiden letzten Unterprogramme gehören zu der Gruppe (D2) von I/O-Programmen. Schließlich sind noch drei Unterprogramme zu nennen, die Dumps der gerade aktuellen Seiten von Namen-, Daten- und Triadenspeicher erzeugen:

(H4) Dump-Programme:

(H4.1) DUMP Namenspeicher : DUMPN(H4.2) DUMP Entityspeicher : DUMPE(H4.3) DUMP triadenspeicher : DUMP

Schließlich wird vom System noch die Existenz einer INTEGER-FUNCTION in der Standard-FØRTRAN-Bibliothek vorausgesetzt, die das bitweise Exklusiv-ØDER (XØR) zweier Maschinenworte bildet. Diese Funktion heißt auf der TR 440 LØGAUT; der amerikanische ISA-Vorschlag, der auf der V73 implementiert ist, nennt die Funktion IEØR. Diese Bezeichnung wird auch in den folgenden Verweislisten immer verwendet; die vorliegende Implementierung auf der TR440 muß natürlich den Namen LØGAUT verwenden.

(FB1) IEØR

2.3.1.4 Die Systemstruktur aus der Sicht des Anwenders

Für den Anwender ergibt sich somit die hier dargestellte Systemstruktur:

<u>Ebene 0</u>	D1: DIALØG				
(Dialog-System)	D3: ALPHAN	(können als Teile von D1 betrachtet werden)			
	ALPHAD				
	NUM				
	G1: STSNGL	G3: DECSET	G5: RDENT	G7: FDELEM	G9: SYSINT
<u>Ebene 1</u>	CHSNGL	STELEM		LIELEM	PAGE
(Unterprogramme)	DLSNGL	DLELEM			SYSGCL
		DLSET			SYSØUT
	G2: CHNAME	G4: DECREL	G6: RDNAME	G8: FDRELE	
	DECSYN	STRELE	RDSYN	FDRTØP	
	DLNAME	DLRELE		FDNEXT	
<u>Ebene 2</u>	S1: STENT	S2: STØRE	S3: NSINIT	S4: ASK	
(Systemdienste)	DELENT	DELETE	F1INIT	ERRØR	
	DSHIFT	RETRIV	HSINIT		
	RTRNID	GARCØL	F2INIT		
			DSINIT		
(Hilfs-routinen)	H1: ALCNAM	H2: KEEPFR	H3: PEØRDR	H4: DUMPN	
	HASHNM	AALLØC	IHASH	DUMPE	
	EXDATA	REQCL1	DINCØD	DUMP	
		PUTRIE	DØUTCD		
		BRSTRT			
		BKTIE			
		F2REL			
		BRETRI			
		CRETRI			
		ALØØK			
		GØBØTM			
	FB: IEØR				

2.3.2 Die Verweisstruktur des Systems

2.3.2.1 Die Verweisebenen

Wenn man die im Kapitel 3 bei den einzelnen Programmen angegebenen Externbezüge auflistet und die Programme nach diesen Bezügen ordnet, erhält man die sogenannte Verweisstruktur des Systems. Diese Struktur erlaubt ebenfalls eine Einteilung der Programme in Ebenen, die Verweisebenen des Systems. Diese Einteilung hat mit der in 2.3.1.4 angegebenen Einteilung nur sehr wenig zu tun, da sie ausschließlich von der logischen Struktur der Programme und nicht von ihrem Einsatzzweck her bestimmt ist. Diese Einteilung in Ebenen ist hier möglich, da aufgrund der Syntax von FØRTRAN keine rekursiven Aufrufe von Unterprogrammen möglich sind - auch nicht mittelbar unter Zwischenschaltung anderer Unterprogramme. In der folgenden Liste wird zu jedem Programm noch die Gruppe angegeben, der es im Abschnitt 2.3.1 aus der Sicht des Anwenders zugeteilt wurde. Auf diese Weise werden Zusammenhänge und Unterschiede zwischen den beiden Strukturen etwas klarer. Das System umfaßt die folgenden 11 Verweisebenen :

Ebene 0 : DIALØG-D1

Ebene 1 : STSNGL,CHSNGL,DLSNGL-G1
 CHNAME,DECSYN,DLNAME-G2
 DECSET,STELEM,DLELEM,DLSET-G3
 DECREL,STRELE,DLRELE-G4
 RDENT-G5
 RDSYN-G6
 FDELEM,LIELEM-G7
 FDRELE,FDRTØP,FDNEXT-G8
 SYSINT,SYSGCL,SYSØUT-G9
 NUM,ALPHAD-D3

Ebene 2 : STØRE,DELETE,RETRIV,GARCØL-S2
 RDNAME-G6
 NSINIT,F1INIT,HSINIT,F2INIT,DSINIT-S3
 ALPHAN-D3

Ebene 3 : KEEPFR,AALLØC,REQCL1,PUTRIE,BRSTRT,BKTIE,F2REL-H2
 EXDATA-H1
 PEØRDR-H3

Ebene 4 : STENT-S1
 IHASH-H3
 BRETRI,CRETRI,ALØØK-H2

 Ebene 5 : ALCNAM-H1
 DELENT-S1
 ASK-S4
 GØBØTM-H2

 Ebene 6 : DSHIFT,RTRNID-S1
 DINCØD,DØUTCD-H3

 Ebene 7 : HASHNM-H1

 Ebene 8 : PAGE-G9
 IEØR-FB

 Ebene 9 : ERRØR-S4

 Ebene 10 : DUMPN,DUMPE,DUMP-H4

Die Routinen STSNGL, CHSNGL, DLSNGL, DLNAME, DECSET wurden der Ebene 1 zugerechnet, obwohl sie im System von keinem Programm aus aufgerufen werden. Eine Zuordnung zur Ebene 0 erschien jedoch nicht sinnvoll, da es sich hier um SUBROUTINES handelt, die ohne aufrufendes Hauptprogramm allein nicht lauffähig sind.

2.3.2.2 Darstellung der Struktur

Auf der nächsten Seite folgt eine graphische Darstellung der Verweisstruktur des Systems. Um wenigstens einen geringen Rest von Übersichtlichkeit wahren zu können, erfolgt die Darstellung etwa in der Form eines elektrischen Leitungsplanes, wo die Verknüpfungsstellen zwischen den einzelnen Leitungen durch Punkte markiert sind. Die Programme sind strikt in ihre Verweisebenen in der Darstellung eingeteilt, so daß alle Verweise von oben nach unten verlaufen. Durch Verfolgen des Weges über die Programme DIALØG,FDRELE,RETRIV,PUTRIE,STENT,DELENT,RTRNID,HASHNM,PAGE,ERRØR,DUMP erkennt man leicht anhand der Darstellung, daß alle 11 Verweisebenen wirklich notwendig sind und daß das System nicht durch eine geringere Anzahl von Ebenen beschrieben werden kann. Dabei spielt es keine Rolle, daß aufgrund der logischen Struktur der einzelnen Programme nie die ganze genannte Folge von Aufrufen durchlaufen werden kann.

3. Beschreibung der Programme

3.1 Vorbemerkung

Die Programme sind nach dem in Abschnitt 2.3.1.4 angegebenen System in dieser Beschreibung angeordnet. Im Quellprogramm selbst sind COMMENT-Zeilen zur leichteren Verständlichkeit überall da eingefügt, wo es notwendig oder sinnvoll erschien; diese Kommentare stehen jeweils hinter der Anweisung bzw. Gruppe von Anweisungen, die durch sie beschrieben wird. Das Vorhandensein dieser Kommentare macht eine Beschreibung der einzelnen Anweisungen an dieser Stelle überflüssig. Die folgenden Programm-Beschreibungen enthalten deshalb lediglich eine globale Beschreibung der Funktionsweise der einzelnen Programme; lediglich bei einigen sehr komplexen Routinen wird eine Aufgliederung angegeben. Der größte Teil der auf den Triadenspeicher arbeitenden Programme der Ebene 2 lehnt sich außerdem in seiner Struktur stark an die in / 6 / angegebenen Flußdiagramme an, so daß aus dieser Schrift weitergehende Informationen über die Struktur eines Teiles der Programme erhalten werden können.

Die Programm-Längen werden in Anzahl der Zeilen des Quelltextes angegeben, da der benötigte Speicherplatz für das übersetzte Programm von Maschine zu Maschine verschieden ist (z.B. auf der Control Data CD3300 werden 21k Kernspeicher benötigt, auf der Telefunken TR440 bei doppelter Wortlänge 29k). Man kann bei den einzelnen Routinen jeweils etwa ein Drittel der angegebenen Zeilenzahl für Kommentarzeilen abrechnen.

3.2 Die Programme der Ebene 0 (Dialogsystem)

3.2.1 Gruppe D1 : PRØGRAM DIALØG

Dieses Programm ist ein Hauptprogramm, das aus Befehlen, die in Form von dreibuchstabigen Mnemocodes mit formatfrei folgenden Parametern bestehen, die gewünschten Unterprogramm-aufrufe erzeugt.

3.2.1.1 Beschreibung des Programms:

Länge: 278

wird aufgerufen von: -

Form des Aufrufs: Das Programm wird als normales Hauptprogramm gestartet.

Parameter: -

Externbezüge: ERROR, DINCØD, NUM, SYSINT, SYSOUT, SYSGCL, ALPHAN, ALPHAD, STENT, STELEM, STRELE, DECSYN, DECREL, DLSET, DELENT, DLRELE, DLELEM, CHNAME, RDENT, DØUTCD, RDSYN, FDELEM, FDRELE, FDRTØP, FDNEXT, LIELEM, DUMPN, DUMP, DUMPE

Common: //IP(2048)/INAME/IN(2048)/IENTI/IE(2048)/BUFFER/IBUF(80),LBUF/STACK/ISTACK(400)/DT/PAGTAB(109)

Fehlerstops: Falls die Bufferlänge (Anzahl der Zeichen, die bei einer Eingabe interpretiert werden) > 80 oder ≤ 30 ist, erfolgt Fehlerstop mit der Meldung:

***** FALSCH E BUFFERLÄNGE

Ausgegebene Texte: * UNBEKANNTE ANWEISUNG (kommt, wenn eine Anweisung eingegeben wird, die nicht aus einem der in 3.2.1.2 angegebenen Mnemocodes besteht)

* FALSCH E ZAHLENEINGABE (bedeutet, daß auf einer Parameterposition, auf der eine Zahl erwartet wird, ein Zeichen eingegeben wurde, das keine Ziffer ist; diese Ausgabe erfolgt speziell auch dann, wenn eine Zahl auf der letzten Parameterposition nicht durch Komma abgeschlossen wurde)

* FALSCH E PARAMETERANZAHL (diese Meldung kommt nur, wenn zuwenig Parameter angegeben wurden; wenn zuviele Parameter angegeben wurden, so werden die restlichen ignoriert)

Diese Fehlermeldungen bewirken sämtlich, daß die letzte Eingabe ignoriert wird. Das Programm meldet sich wieder und verlangt eine neue Eingabe.

===== BITTE BUFFERLÄNGE EINGEBEN

Dies ist die erste Ausgabe vom Programm; es wartet dann, bis eine Zahl zwischen 31 und 80 im FØRMAT(I2) eingegeben wird.

===== BITTE ANWEISUNG EINGEBEN (Nur nach Ausgabe dieses Textes kann ein Befehl eingegeben werden.)

DATENLAENGE =, KLASSE = (gibt beim Befehl RDE die Länge des Datenblocks in Character und die Klasse der gelesenen Entity an. In den folgenden Zeilen wird der gewählte Datenbereich der Entity formatfrei ausgegeben.)

SYN.ZAHL =, SYN.NR. = (gibt beim Befehl RDS die Gesamtzahl der Synonyme und die Nummer des ausgegebenen Synonyms an.)

TRIADENSEITE =, INDEX = (gibt beim Befehl FDN die Triadenspeicher-Adresse des nächsten Elementes auf dem Ring an und beim Befehl FDR die Adresse des gefundenen Elementes.)

ANZAHL DER ELEMENTE = (gibt beim Befehl LIE die Mächtigkeit der angesprochenen Menge an.)

Programmaufbau: Nach dem Einlesen der Bufferlänge LBUF tritt das Programm in eine endlose Schleife ein, die nur durch Eingabe des Befehls SYØ wieder verlassen werden kann. In dieser Schleife wird nach Ausgabe des Textes

===== BITTE ANWEISUNG EINGEBEN

ein Datensatz von genau LBUF Characters im FØRMAT(80A1) eingelesen und von der Routine DINCØD umcodiert. Dann wird abgefragt, zu welcher der in 3.2.1.3 angegebenen Gruppen der eingegebene Befehl gehört, und innerhalb der gefundenen Gruppe wird noch einmal abgefragt, welcher Befehl dieser Gruppe vorliegt. Bei einem fehlerhaften Mnemocode wird an den Anfang der Schleife zurückgesprungen. Als nächstes erfolgt ein Abchecken auf die richtige Parameteranzahl; Fehler führen wieder zum Rücksprung zum Beginn der Schleife. Nach dem Herstellen der geeigneten Parameter mittels der Routinen ALPHAN, ALPHAD und NUM und gegebenenfalls noch einer Überprüfung der Korrektheit einer Zahleneingabe wird die

gewünschte Routine aufgerufen. Falls dann noch Informationen vorliegen, die von dieser Routine nicht ausgegeben wurden, so erfolgt eine Ausgabe dieser Informationen. Anschließend erfolgt beim Befehl SYØ eine Beendigung des Programmlaufes, bei allen anderen Befehlen ein Rücksprung zum Beginn der Schleife.

3.2.1.2 Ablauf des Dialogs

Nach dem Start des Programms kommt als erstes die Meldung:

===== BITTE BUFFERLAENGE EINGEBEN

Die Bufferlänge ist im FØRMAT(I2) einzugeben. Falls dabei ein Fehler gemacht wird, so wird die Meldung wiederholt und eine neue Eingabe erwartet; falls aber eine Bufferlänge ≤ 30 oder >80 angegeben wird, erfolgt ein Fehlerstop. Die weiteren Eingaben geschehen zeilenweise auf Aufforderung vom System hin. Die ersten drei Zeichen einer Zeile werden als Befehlscode interpretiert; die hier erlaubten Codes sind im Abschnitt 3.2.1.3 angegeben. Ein unerlaubter Befehlscode führt, ebenso wie die Eingabe eines Zeichens, das nicht zum erkennbaren Zeichenvorrat

_(blank) 0 ... 9 A ... Z + - * / . () , = ' _

gehört, oder auch wie ein Fehler bei einer Zahleneingabe oder eine zu geringe Parameteranzahl zu einer Fehlermeldung und zum Ignorieren dieser Eingabe. Falls Parameter angegeben werden müssen, so folgen sie formatfrei auf den Befehlscode, jeweils durch Kommata abgetrennt. Wenn der letzte Parameter eine Zahl ist, muß ein Komma folgen; andernfalls darf ein Komma oder Apostroph folgen, es muß aber nichts folgen. Das Ende einer Dateneingabe (bei einem der Befehle STS, STE oder CHS), die sich über mehrere Zeilen erstrecken darf, muß durch ein Apostroph angezeigt werden. Ein Apostroph in einer Zeile bewirkt den Abbruch des Einlesens der Zeile an dieser Stelle; ein Komma in einer Befehlszeile trennt Parameter voneinander ab.

Wird nicht als erster Befehl der Befehl SYI gegeben, so ver-

sucht das Programm einen Restart mit Informationen, die auf einem externen Speichermedium mit der logischen Gerätenummer 10 stehen. Speziell wird von dort die für diese Informationen gültige Page-Tabelle eingelesen.

3.2.1.3 Befehlsvorrat

3.2.1.3.1 SYSTEMBEFEHLE

SYI Initialisierung : SYI, $n_n, n_t, n_d, n_r, n_h, n_c,$

bewirkt: CALL SYSINT(NN, NT, ND, NR, NH, NC, LBUF)

n_n, n_t, n_d : Anzahl der Seiten für Namen-, Triaden- und Datenspeicher. Es muß (bei der vorliegenden Implementierung) gelten : $n_n + n_t + n_d < 100$

n_r : Anzahl der A-Zellen in F1, d.h. Anzahl der auf einer Triadenseite abzuspeichernde Relationen; $n_r < 100$ ist empfehlenswert

n_h : Anzahl der B-Zellen in der Hash-Area des Triadenspeichers; $100 < n_h < 200$ ist empfehlenswert

n_c : Anzahl der Zellen in der Hash-Area des Namenspeichers; $100 < n_c < 200$ ist empfehlenswert.

Nach / 5 / empfiehlt es sich weiterhin, für n_h und n_c Primzahlen zu wählen.

SYG Garbage-Collection des Triadenspeichers : SYG

bewirkt: CALL SYSGCL

SYØ System-Øut : SYØ

bewirkt: CALL SYSØUT und beendet die Kommunikation mit dem System.

3.2.1.3.2 STØRE-BEFEHLE

STS Single : STS, name, daten'

bewirkt : CALL STSNGL (NAME, IDATLG, IDATA)

Die Daten können sich über mehrere Zeilen erstrecken und müssen mit Apostroph abgeschlossen sein.

STE Element: STE,name,namecl,daten'

bewirkt : CALL STELEM(NAME,IDATLG,IDATA,NAMECL)

Dabei ist "name" der Name des Elements, "namecl" der Name der Menge.

STR Relation: STR,relation "A",entry "B",entry "C" permutations-code,

bewirkt : CALL STRELE(NAMEA,NAMEB,NAMEC,IPERMT)

3.2.1.3.3 DECLARE-BEFEHLE

DCS Synonym : DCS,name neu,name alt

bewirkt : CALL DECSYN(NAMNEU,NAMALT)

"name alt" wird auch unter "name neu" ansprechbar.

DCC Class : DCC,namecl

bewirkt : CALL DECSET(NAMECL)

DCR Relation : DCR,name,permutations-code,

bewirkt : CALL DECREL(NAME,IPERMT)

3.2.1.3.4 DELETE-BEFEHLE

DLC Class : DLC,namecl

bewirkt : CALL DLSET(NAMECL)

Es werden von den Elementen nur die Single-Value-Entities gelöscht.

DLN Name: DLN,name

bewirkt : CALL DLNAME(NAME)

DLS Single: DLS,name

bewirkt: CALL DLSNGL(NAME)

DLR Relation: DLR,relation "A",entry "B",entry "C",code,

bewirkt: CALL DLRELE(NAMEA,NAMEB,NAMEC,MØD)

code = 0 : nur der Relationszusammenhang (die Triade)
wird gelöscht.
= 1 : außerdem wird die Entity C gelöscht.
= 2 : außerdem werden B und C gelöscht.
= 3 : außerdem werden A, B und C gelöscht.

DLE Element: DLE, name, namecl
bewirkt : CALL DLELEM(NAME, NAMECL)

3.2.1.3.5 CHANGE-BEFEHLE

CHS Single : CHS, name, daten'
bewirkt : CALL CHSNGL(NAME, IDATLG, IDATA)
Die Daten werden unter Beibehaltung des Namens geändert.

CHN Name : CHN, name neu, name alt
bewirkt : CALL CHNAME(NAMNEU, NAMALT)
"name alt" wird unter Beibehaltung der Daten in "name neu"
umbenannt.

3.2.1.3.6 READ-BEFEHLE

RDE Entity : RDE, name, nummer,
bewirkt : CALL RDENT(NAME, IDATLG, ITYPE, NUMMER)
der Datenbereich mit der Nummer "nummer" wird bis zu
einer maximalen Länge von 400 Character ausgegeben.

RDS Synonym : RDS, name, nummer,
bewirkt : CALL RDSYN(NAME, NAMES, NUMMER, ISYN, ICØD)
Das Synonym von "name" mit der Nummer "nummer" wird ge-
lesen.

3.2.1.3.7 FIND-BEFEHLE

FDE Element : FDE, name, namecl
bewirkt : CALL FDELEM(NAME, NAMECL, ICØD)
Es wird festgestellt, ob eine Elementbeziehung besteht.

FDR Relation : FDR,relation "A",entry "B",entry "C"
 bewirkt : CALL FDRELE(NAMEA,NAMEB,NAMEC,IPAGET,
 IND,ICØD)

Es wird festgestellt, ob der Relationszusammenhang besteht. Jeder der Parameter kann weggelassen werden; dabei muß das zugehörige Komma aber geschrieben werden. In diesem Fall wird gesucht, ob ein Zusammenhang zwischen den restlichen Parametern besteht. Wurde nur ein Parameter weggelassen, so wird als Antwort der Name der Entity übergeben, die den Relationszusammenhang erfüllt. Falls es mehrere Antworten gab, so wird mit der Routine PUTRIE eine Multi-Value-Entity erzeugt, die die Antworten als Elemente enthält, und deren Name als Antwort übergeben. Wurden zwei Parameter weggelassen, so werden die Indizes des Top des Ringes übergeben, der die Antworten enthält.

FDT Ring-Top : FDT bzw. FDT,seitennummer,index,
 bewirkt: CALL FDRTØP(NAME,IPAGET,IND,ICØD)
 Es wird der Name der Entity übergeben, auf die die Seitennummer im Triadenspeicher und der Index auf der Seite zeigen. Falls diese Parameter nicht angegeben werden, so werden die Werte benutzt, auf denen die entsprechenden Variablen zur Zeit stehen.

FDN Next : FDN bzw. FDN,seitennummer,index,
 bewirkt : CALL FDNEXT(NAME,IPAGET,IND,ICØD)
 Es wird der Name der nächsten Entity auf dem Ring übergeben sowie deren Seitennummer und Index auf der Seite im Triadenspeicher. Die Parameter können wieder weggelassen werden.

3.2.1.3.8 LIST-BEFEHL

LIE Element : LIE,namecl,nummer,
 bewirkt : CALL LIELEM(NAME,NAMECL,NR,ID1,ID2)
 Der Name des Elements mit der Nummer "nummer" wird ausgegeben.

3.2.1.3.9 DUMP-BEFEHLE

DUN Namenspeicher : DUN
bewirkt : CALL DUMPN

DUT Triadenspeicher : DUT
bewirkt : CALL DUMP

DUE Entityspeicher : DUE
bewirkt : CALL DUMPE

Die jeweils aktuelle Seite wird auf dem Drucker angegeben.

3.2.2 Gruppe D3

3.2.2.1 SUBROUTINE ALPHAN(INW,INB,IKT,NAME)

Länge : 19

wird aufgerufen von : DIALØG, ALPHAD

Form des Aufrufs: INTEGER NAME(INW),IKT(25)
CALL ALPHAN(INW,INB,IKT,NAME)

Eingangsparameter : INW : Länge(in Worten)des zu erzeugenden
Namens
INB : Nummer des Blocks in der Eingabezeile,
der zu konvertieren ist
IKT : von DINCØD erzeugtes Feld, das die
Einteilung der Eingabezeile enthält

Ausgangsparameter : NAME : Feld, auf das der erzeugte Name
geschrieben wird.

Transients : -

Externbezüge : -

Common : /BUFFER/IBUF(80),LBUF

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Nachdem zunächst das Feld NAME gelöscht, d.h. mit Blanks vorbesetzt wurde, und die Länge des aktuellen Namens bestimmt wurde, werden die geeigneten Elemente des Common-Feldes IBUF zu je zweien in ein Element des Feldes NAME übertragen nach der Formel:

$$\text{NAME(I)} = 256 \cdot \text{IBUF(J-1)} + \text{IBUF(J)}$$

Der Name steht nach korrekter Konvertierung linksbündig im Feld NAME. Hatte der konvertierte Block der Eingabezeile mehr als 2*INW Characters, so werden die restlichen Characters dieses Blocks überlesen. Eine Fehlermeldung erfolgt in diesem Falle nicht.

3.2.2.2 SUBROUTINE ALPHAD(IDATLG,INB,IKT,IFELD)

Länge : 58

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER IKT(25),IFELD(40)

CALL ALPHAD(IDATLG,INB,IKT,IFELD)

Eingangsparameter : INB : Nummer des Blocks in der Befehlszeile der Eingabe, der zu konvertieren ist

Ausgangsparameter : IDATLG : Länge (in Worten) des konvertierten Datenblocks

IFELD : Hilfsfeld, das benötigt wird, um eine umcodierte Zeile von DINCØD an die geeigneten Elemente von ISTACK zu übergeben

Transients : IKT : von DINCØD erzeugtes Feld, das die Einteilung der Eingabezeile enthält

Externbezüge : ALPHAN,DINCØD

Common : /BUFFER/IBUF(80),LBUF/STACK/ISTACK(400)

Fehlerstops : -

Ausgegebene Texte : Wenn sich die zu konvertierenden Daten über mehrere Zeilen erstrecken, werden neue Zeilen mit dem Text

===== BITTE NAECHSTEN DATENBLOCK EINGEBEN

angefordert. Wenn dieser nächste Datenblock nicht mehr voll übertragen werden kann (weil sonst das Maximum von 800 Characters überschritten wird), so wird er mit dem Text

===== BITTE LETZTEN DATENBLØCK EINGEBEN

===== MAX. LAENGE =

angefordert. Ist die Eingabe länger als der noch zu übertragende Bereich, dann werden die restlichen Characters dieser Zeile überlesen. Eine Fehlermeldung erfolgt in diesem Falle nicht.

Programmaufbau : Nachdem zunächst das Feld ISTACK gelöscht, d.h. mit Blanks vorbesetzt wurde, wird abgefragt, ob die Daten sich über mehrere Zeilen erstrecken. Wenn das nicht der Fall ist, so werden die Daten durch Aufrufen von ALPHAN konvertiert, und es erfolgt ein Rücksprung ins aufrufende Programm. Andernfalls wird die erste Zeile ebenfalls durch Aufrufen von ALPHAN konvertiert, und es erfolgt Eintritt in eine Schleife, die mit der Abfrage beginnt, ob noch eine volle Zeile übertragen werden kann. Wenn das der Fall ist, wird die Schleife durchlaufen; es wird eine neue Eingabezeile angefordert, gelesen und durch Aufruf von DINCØD konvertiert. Dabei wird abgecheckt, ob ein Zeichen eingegeben wurde, das nicht zum Zeichenvorrat gehört - in diesem Fall wird zum Beginn der Schleife gesprungen - und ob das Ende der zu übertragenden Daten erreicht wurde - in diesem Fall erfolgt nach dem Übertragen der

Daten ein Rücksprung ins aufrufende Programm. Die Daten, die jetzt noch auf dem Feld IFELD stehen, werden auf die geeigneten Elemente von ISTACK übertragen, und es erfolgt Rücksprung zum Beginn der Schleife. Falls keine ganze Zeile mehr übertragen werden kann, so wird nach Ausgabe der Meldung und Umcodieren der Zeile durch DINCØD die Marke für das Ende des Datenblocks an die entsprechende Stelle der Zeile geschrieben, und es erfolgt ein Sprung in die Schleife. Die Daten stehen nach der Konvertierung linksbündig im Feld ISTACK.

3.2.2.3 SUBROUTINE NUM(INW,INB,IKT,IRC)

Länge : 22

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER IKT(25)

CALL NUM(INW,INB,IKT,IRC)

INW : Anzahl der zu konvertierenden Zahlen

INB : Nummer des ersten Blocks in der Eingabezeile, der zu konvertieren ist

IKT : von DINCØD erzeugtes Feld, das die Einteilung der Eingabezeile enthält

Ausgangsparameter : IRC : = 0, wenn alle Zahlen ordnungsgemäß konvertiert wurden

= 1, wenn ein Fehler auftrat, speziell wenn eines der Zeichen der Eingabezeile keine Ziffer war oder wenn eine Zahl nicht durch Komma abgeschlossen wurde

Transients : -

Externbezüge : -

Common : /BUFFER/IBUF(80),LBUF/STACK/ISTACK(24)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Das Programm ist in Form einer doppelten DØ-Schleife aufgebaut. In der inneren Schleife werden die geeigneten, in IBUF stehenden Ziffern nach dem Horner-Schema zu einer Dezimalzahl zusammengefügt und in ein Element des Feldes ISTACK übertragen. Die äußere Schleife bestimmt für jede Zahl Anfang und Ende aus der Information des Feldes IKT. Das Ergebnis steht nach korrekter Konvertierung in den ersten INW Elementen von ISTACK. Wenn ein Fehler auftrat, wird die Konvertierung an dieser Stelle abgebrochen und der Reject-Code IRC=1 zurückgegeben.

3.3 Die Programme der Ebene 1 (Unterprogramme)

3.3.1 Gruppe G1 (Manipulation von Single-Value-Entities)

3.3.1.1 SUBROUTINE STSNGL(NAME,IDATLG,IDATA)-Store Single

Länge : 5

wird aufgerufen von : -

Form des Aufrufs : INTEGER NAME(5),IDATA(IDATLG)
CALL STSNGL(NAME,IDATLG,IDATA)

Eingangsparameter : NAME : Name der Entity, max. 10 Characters,
die linksbündig stehen müssen. Ein
Name, der mit zwei Blanks anfängt, ist
nicht erlaubt.

IDATLG:Länge des Datenblocks (in Worten)

IDATA: Feld, das die Daten enthält

Ausgangsparameter : -

Transients : -

Externbezüge : STENT

Common : -

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Das Programm ruft STENT mit den richtigen Parametern auf. Es dient nur zur Einsparung von Parametern und zur Bequemlichkeit des Anwenders.

3.3.1.2 SUBROUTINE CHSNGL(NAME,IDATLG,IDATA)-Change Single

Länge : 6

wird aufgerufen von : -

Form des Aufrufs : INTEGER NAME(5),IDATA(IDATLG)
CALL CHSNGL(NAME,IDATLG,IDATA)

Eingangsparameter : NAME : Name der Entity (siehe 3.3.1.1)
IDATLG: Länge des Datenblocks (in Worten)
IDATA : Feld, das die Daten enthält

Ausgangsparameter : -

Transients : -

Externbezüge : DELENT,STENT

Common : -

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Das Programm ruft zuerst DELENT, dann STENT mit den richtigen Parametern auf. Es dient nur zur Einsparung von Parametern und zur Bequemlichkeit des Anwenders, speziell zur Vermeidung einer Routinefrage an den Anwender, die bei direktem Aufruf von STENT gestellt würde.

3.3.1.3 SUBROUTINE DLSNGL(NAME)-Delete Single

Länge : 5

wird aufgerufen von : -

Form des Aufrufs : INTEGER NAME(5)

CALL DLSNGL(NAME)

Eingangsparameter : NAME : Name der Entity (siehe 3.3.1.1)

Ausgangsparameter : -

Transients : -

Externbezüge : DELENT

Common : -

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Das Programm ruft DELENT mit den richtigen Parametern auf. Es dient nur zur Einsparung von Parametern und zur Bequemlichkeit des Anwenders.

3.3.2 Gruppe G2(Manipulation von Namen)

3.3.2.1 SUBROUTINE CHNAME(NAMNEU,NAMALT)-Change Name

Länge : 22

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAMNEU(5),NAMALT(5)

CALL CHNAME(NAMNEU,NAMALT)

Eingangsparameter : NAMNEU : neuer Name der Entity

NAMALT : alter, zu löschender Name der Entity

Ausgangsparameter : -

Transients : -

Externbezüge : RTRNID,ERRØR,ASK,DELENT,STENT

Common : -

Fehlerstops : -

Ausgegebene Texte : Wenn die Entity NAMALT nicht existiert, erfolgt die Meldung
 *****KEINE ENTITY DIESES NAMENS DA
 und Rücksprung ins aufrufende Programm.
 Existiert schon eine Entity NAMNEU, so wird der Anwender gefragt :
 ----- UEBERSPEICHERN VON
 ----- name neu
 Vom Anwender wird eine Antwort erwartet.
 Sind deren erste beide Buchstaben JA oder ØK, so wird der Name ausgetauscht, nachdem vorher die Entity NAMNEU gelöscht wurde, andernfalls erfolgt ein Rücksprung ins aufrufende Programm.

Programmaufbau : Nachdem durch RTRNID festgestellt wurde, ob NAMALT und/oder NAMNEU existieren, und nachdem gegebenenfalls die Erlaubnis eingeholt wurde, NAMNEU zu löschen, und dies durch Aufruf von DELENT geschehen ist, wird durch Aufruf von STENT mit NAMNEU und dem durch RTRNID bestimmten ID von NAMALT als Parameter der neue Name dieser Entity zugeordnet. Durch anschließenden Aufruf von DELENT mit den geeigneten Parametern wird dann der Name NAMALT gelöscht. Die Entity ist nach dem Aufruf von CHNAME nicht mehr unter NAMALT sondern nur noch unter NAMNEU ansprechbar.

3.3.2.2 SUBROUTINE DECSYN(NAMNEU,NAMALT)-Declare Synonym

Länge : 20

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAMNEU(5),NAMALT(5)
 CALL DECSYN(NAMNEU,NAMALT)

Eingangsparameter : NAMNEU : neuer Name der Entity
 NAMALT : alter Name der Entity

Ausgangsparameter : -

Transients : -

Externbezüge : RTRNID,ERRØR,ASK,DELENT,STENT

Common : -

Fehlerstops : -

Ausgegebene Texte : Wenn die Entity NAMALT nicht existiert,
 erfolgt die Meldung
 ***** KEINE ENTITY DIESES NAMENS DA
 und Rücksprung ins aufrufende Programm.
 Existiert schon eine Entity NAMNEU, so
 wird der Anwender gefragt
 ----- UEBERSPEICHERN VØN
 ----- name neu
 Sind die beiden ersten Buchstaben der Ant-
 wort des Anwenders JA oder ØK, so wird die
 Entity NAMNEU gelöscht und der Name NAMNEU
 zu einem Synonym der Entity NAMALT gemacht,
 andernfalls erfolgt ein Rücksprung ins
 aufrufende Programm.

Programmaufbau : Nachdem durch RTRNID festgestellt wurde, ob
 NAMALT und/oder NAMNEU existieren, und nachdem
 gegebenenfalls die Erlaubnis eingeholt wurde,
 NAMNEU zu löschen und dies durch Aufruf von
 DELENT geschehen ist, wird durch Aufruf von STENT
 mit dem ID von NAMALT als Parameter der Name
 NAMNEU der Entity NAMALT zugeordnet. Diese Entity
 ist jetzt unter beiden Namen ansprechbar.

3.3.2.3 SUBROUTINE DLNAME(NAME)-Delete Name

Länge : 5

wird aufgerufen von : -

Form des Aufrufs : INTEGER NAME(5)
 CALL DLNAME(NAME)

Eingangsparameter : NAME : Name der Entity, der gelöscht werden soll

Ausgangsparameter : -

Transients : -

Externbezüge : DELENT

Common : -

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Das Programm ruft DELENT mit den richtigen Parametern auf. Es dient nur zur Einsparung von Parametern und zur Bequemlichkeit des Anwenders.

3.3.3 Gruppe G3 (Manipulation von Mengen)

3.3.3.1 SUBROUTINE DECSET(NAME)-Declare Set (Declare Class)

Länge : 6

wird aufgerufen von : -

Form des Aufrufs : INTEGER NAME(5)
CALL DECSET(NAME)

Eingangsparameter : NAME : Name der Menge

Ausgangsparameter : -

Transients : -

Externbezüge : STENT

Common : -

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Das Programm ruft STENT mit den richtigen Parametern auf. Es dient nur zur Einsparung von Parametern und zur Bequemlichkeit des Anwenders.

3.3.3.2 SUBROUTINE STELEM(NAME, IDATA, NAMECL) - Store Element

Länge : 58

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAME(5), NAMECL(5), IDATA(IDATLG), PAGTAB(109)
CALL STELEM(NAME, IDATLG, IDATA, NAMECL)

Eingangsparameter : NAME : Name des Elements

IDATLG : Länge (in Worten) des Datenblocks
des Elements

IDATA : Feld, das die Daten des Elements
enthält

NAMECL : Name der Menge

Ausgangsparameter : -

Transients : -

Externbezüge : RTRNID, STENT, PAGE, ERRØR, DSHIFT

Common : /IENTI/IE(2048)/DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : Wenn NAMECL schon vorhanden ist und zur
Klasse der Relation-Entities gehört, erfolgt
die Meldung

***** RELATIØN KANN NICHT MENGE SEIN

und Rücksprung ins aufrufende Programm. Wenn
auf der Seite des Datenspeichers, auf der die
Menge NAMECL steht, nicht mehr genügend Platz
für die Eintragung der Elementbeziehung ist,
erfolgt die Meldung

***** UEBERLAUF BEI ELEMENT-SPEICHERUNG

und Rücksprung ins aufrufende Programm.

Programmaufbau : Mittels RTRNID wird festgestellt, ob das Element
schon existiert; falls nicht, wird es mittels
STENT erzeugt. Dann wird mittels RTRNID festge-
stellt, ob die Menge schon existiert; falls
nicht, wird sie als einelementige Menge mittels
STENT erzeugt, und es erfolgt ein Rücksprung ins
aufrufende Programm. Andernfalls wird festge-

stellt, zu welcher Entity-Klasse NAMECL gehört. Wenn NAMECL eine Relation-Entity ist, erfolgt eine Fehlermeldung und ein Rücksprung ins aufrufende Programm. Wenn NAMECL dagegen eine Single-Value-Entity ist, wird die Entity dazu zu einer Multi-Value-Entity undefiniert. Durch Verschieben der Daten auf der Datenspeiche von NAMECL wird Platz für die Eintragung der Elementbeziehung geschaffen. Weiterhin wird die Page-Tabelle aktualisiert. Schließlich wird der Elementzähler der Menge aktualisiert und die Elementbeziehung eingetragen. Ein Abchecken, ob diese Beziehung schon bestand, erfolgt nicht. Der Anwender kann dies jedoch selbst durch Aufrufen von FDELEM machen, wenn es nötig erscheint, um Doppeleintragungen zu vermeiden.

3.3.3.3 SUBROUTINE DLELEM(NAME,NAMECL)-Delete Element

Länge : 48

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAME(5),NAMECL(5),PAGTAB(109)
CALL DLELEM(NAME,NAMECL)

Eingangsparameter : NAME : Name des Elementes
NAMECL : Name der Menge

Ausgangsparameter : -

Transients : -

Externbezüge : RTRNID,ERRØR,PAGE,DSHIFT,DELENT

Common : /IENTI/IE(2048)/DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : Wenn die Entity NAMECL nicht existiert, folgt die Meldung
***** MENGE NICHT VORHANDEN
und Rücksprung ins aufrufende Programm.
Wenn NAMECL keine Menge ist, erfolgt die

Meldung

***** KEINE MULTI-VALUE-ENTITY

und Rücksprung. Wenn NAMECL keine Elemente
bzw. nicht NAME als Element hat, erfolgt die
Meldung

***** LEERE MENGE

bzw.

***** KEINE ELEMENTBEZIEHUNG

und Rücksprung.

Programmaufbau : Durch Aufruf von RTRNID wird festgestellt,
ob das Element NAME existiert, falls nicht,
erfolgt ohne Meldung ein Rücksprung ins aufrufende Programm. Nachdem dann die anderen
Fehlermöglichkeiten in der oben angegebenen
Reihenfolge abgefragt wurden, wird durch geeigneten Aufruf von DSHIFT die Eintragung der
Elementbeziehung in NAMECL gelöscht. Anschließend
wird der Elementzähler von NAMECL aktualisiert
und schließlich durch Aufruf von DELENT das
Element gelöscht.

3.3.3.4 SUBROUTINE DLSET(NAME)-Delete Set (Delete Class)

Länge : 73

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAME(5), PAGTAB(109)
CALL DLSET(NAME)

Eingangsparameter : NAME : Name der Menge

Ausgangsparameter : -

Transients : -

Externbezüge : RTRNID, PAGE, ERRØR, DSHIFT, DELENT, ASK

Common : /IENTI/IE(2048)/STACK/ISTACK(200)/DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : Wenn NAME keine Menge ist, erfolgt die Meldung

***** KEINE MULTI-VALUE-ENTITY

und es erfolgt ein Rücksprung ins aufrufende Programm. Wenn die Menge mehr als 100 Elemente hat, werden nur die ersten 100 gelöscht, und die Menge existiert weiter, allerdings eben mit geringerer Mächtigkeit. Dieser Tatbestand wird durch die Meldung

***** NUR 100 ELEMENTE GELØESCHT

kenntlich gemacht. Soll ein Element gelöscht werden, das durch einen Fehler des Anwenders schon gelöscht wurde, ohne daß die Eintragung der Elementbeziehung gelöscht wurde, so erfolgt die Meldung

***** UNGUELTIGES ID

Befinden sich schließlich unter den Elementen Relationen und Mengen, so werden diese nicht gelöscht, und ihre Namen werden mit folgender Meldung ausgegeben :

----- KEIN LØESCHEN VØN

-----

In dem Fall, daß NAME mehr als 100 Elemente besitzt, wird somit bei Relationen und Mengen nur die Eintragung der Elementbeziehung gelöscht.

Programmaufbau : Durch Aufruf von RTRNID wird festgestellt, ob die Entity NAME existiert; falls nicht, erfolgt ohne Meldung ein Rücksprung ins aufrufende Programm. Nachdem abgefragt wurde, ob NAME eine Mult Value-Entity ist, werden die IDs der Elemente (max. 100) in das Hilfsfeld ISTACK übertragen. Falls nun die Mächtigkeit der Menge > 100 ist, werden die ersten 100 Eintragungen von Elementbeziehungen gelöscht; andernfalls wird die gesamte Entity NAME gelöscht. Schließlich werden in einer DØ-Schleife für jedes der IDs, die im Feld ISTACK gespeichert sind, der Name der zu-

gehörigen Entity und ihre Klasse bestimmt. Falls es sich um eine Single-Value-Entity handelt, wird sie dann gelöscht, andernfalls erfolgt nur eine Meldung, daß sie nicht gelöscht wurde. Das Programm ist so ausgelegt, daß durch mehrfachen Aufruf hintereinander beliebig mächtige Mengen gelöscht werden können.

3.3.4 Gruppe G4 (Manipulation von Relationen)

3.3.4.1 SUBROUTINE DECREL(NAME,IPERMT)-Declare Relation

Länge : 8

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAME(5),PAGTAB(109)
CALL DECREL(NAME,IPERMT)

Eingangsparameter : NAME : Name der Relation

IPERMT : Permutationscode der Relation

(siehe Abschnitt 2.2.2.3 und 3.3.4.2)

Ausgangsparameter : -

Transients : -

Externbezüge : STENT

Common : /DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Das Programm ruft STENT mit den richtigen Parametern auf. Es dient nur zur Einsparung von Parametern und zur Bequemlichkeit des Anwenders.

3.3.4.2 SUBROUTINE STRELE (NAMEA, NAMEB, NAMEC, IPERMT) - Store Relation Entry

Länge : 147

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAMEA(5), NAMEB(5), PAGTAB(109)
CALL STRELE (NAMEA, NAMEB, NAMEC, IPERMT)

Eingangsparameter : NAMEA : Name "A" der Relation
NAMEB : Name "B" des ersten Entry
NAMEC : Name "C" des zweiten Entry
IPERMT : Permutationscode (muß angegeben werden, wird aber nur berücksichtigt, wenn "A" noch nicht existiert)
= 0 : nur die Permutation ABC wird abgespeichert
= 1 : die Permutationen ABC und BCA werden abgespeichert
= 2 : die Permutationen ABC, BCA und CAB werden abgespeichert

Ausgangsparameter : -

Transients : -

Externbezüge : RTRNID, STENT, ERRØR, PAGE, DSHIFT, STØRE

Common : //IP(2048)/IENTI/IE(2048)/DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : Wenn für die Relation ein Name angegeben wird, der mit zwei Blanks beginnt, erfolgt die Meldung
***** ILLEGALER RELATIØNS-NAME
und Rücksprung ins aufrufende Programm.
Wenn die Entity NAMEA schon existiert und vom Typ Multi-Value-Entity ist, erfolgt die Meldung
***** MENGE KANN NICHT RELATIØN SEIN
und Rücksprung. Die Entities NAMEB und NAMEC

wurden in diesen beiden Fällen, soweit sie nicht schon existierten, erzeugt. Bei allen anderen Fehlermeldungen wurde außerdem die Entity NAMEA erzeugt bzw. undefiniert, falls das notwendig war, und erst dann erfolgen die folgenden Meldungen mit anschließendem Rücksprung. Wenn auf der Seite des Datenspeichers, auf der die Entity NAMEA steht, kein Platz mehr ist für die Eintragung der Seite des Triadenspeichers, erfolgt die Meldung

***** UEBERLAUF BEI RELATION-SPEICHERUNG

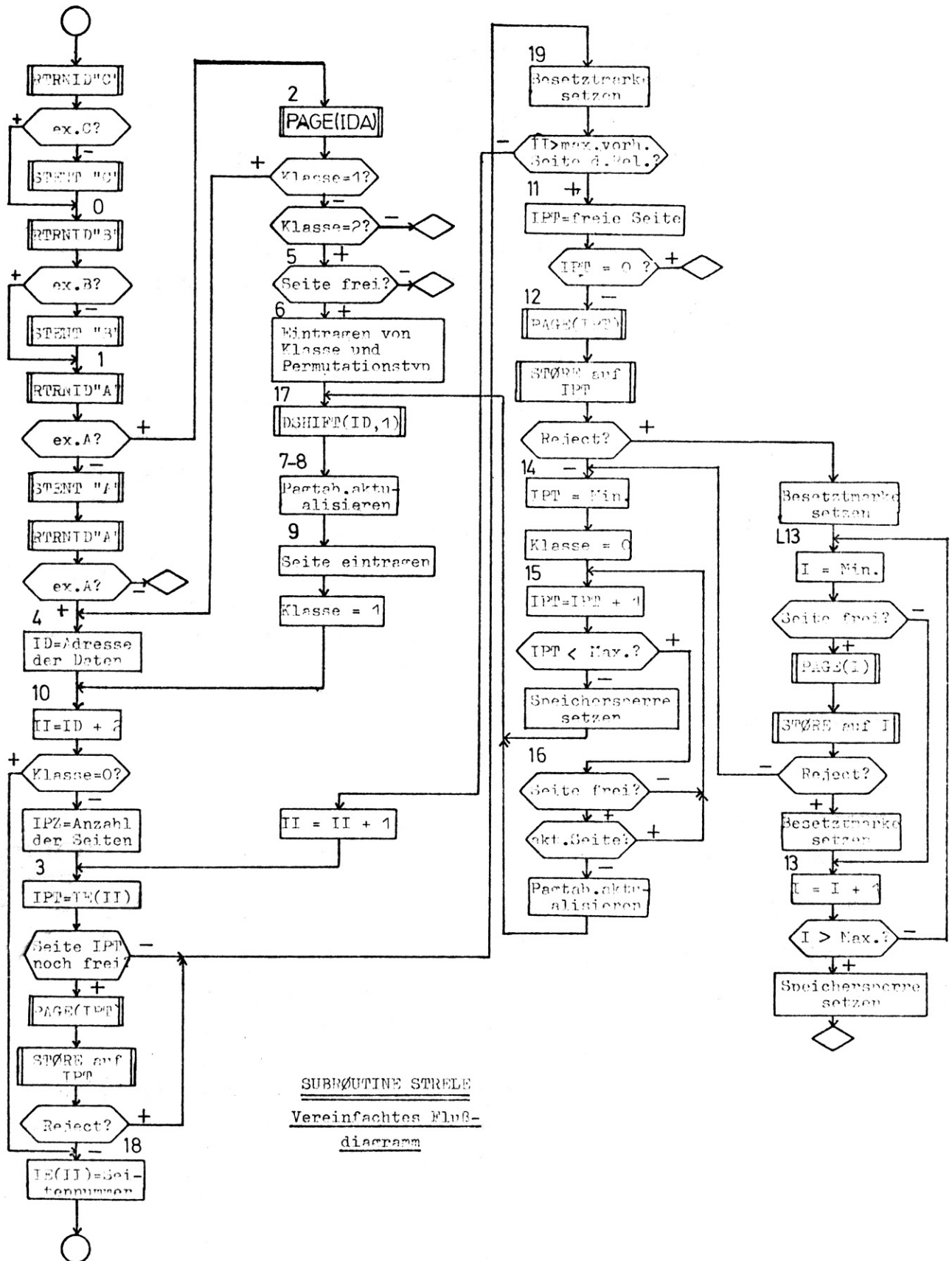
Wenn sich auf keiner der Seiten des Triadenspeichers mehr alle geforderten Permutationen der abzuspeichernden Triade unterbringen lassen, erfolgt die Meldung

***** TRIADENSPEICHER VÖLL

Dabei können auf einigen Seiten des Triadenspeichers Abspeicherversuche vorgekommen sein, die jedoch nicht alle Permutationen erfassen konnten. Diese Triaden sind für das Dialog-System verloren; sie können jedoch durch Aufruf von PAGE und den geeigneten Triadenspeicher-Programmen noch verarbeitet werden.

Programmaufbau : Zunächst wird abgefragt, ob NAMEC, NAMEB und NAMEA existieren; nicht existierende Entities werden durch Aufruf von STENT erzeugt, und zwar NAMEC und NAMEB als Single-Value-Entities, NAMEA als Relation-Entity. Dabei wird aus Sicherheitsgründen abgefragt, ob NAMEA ein erlaubter Name ist. Wenn NAMEA schon als Single-Value-Entity existiert, so wird diese, unter Beibehaltung ihrer Daten, zu einer Relation-Entity umdefiniert. Es erfolgt dann ein Versuch, alle Permutationen der Triaden mittels STØRE auf der in PAGTAB(5) vorgemerkten freien Seite des Triadenspeichers abzuspeichern, oder, falls schon Triaden der Relation

NAMEA existieren, auf einer der Seiten, auf denen diese stehen. Wenn dieser Abspeicherversuch erfolgreich verläuft, d.h. wenn von STØRE kein Reject-Code zurückgegeben wird, wird die Seite des Triadenspeichers, soweit das noch nicht geschehen ist, in die Entity NAMEA eingetragen, und es erfolgt ein Rücksprung ins aufrufende Programm. Kommt jedoch ein Reject-Code zurück, so wird zunächst versucht, die Permutationen der Triade auf den noch freien Seiten des Triadenspeichers unterzubringen, die schon Triaden der Relation NAMEA enthalten. Dabei werden Seiten, die zu einem Reject-Code geführt haben, als besetzt markiert. Da diese Markierung in der Page-Tabelle geschieht, werden besetzte Seiten also gar nicht erst in den Kernspeicher transportiert. Hat auch dieses Verfahren noch nicht zu einem Erfolg geführt, so wird in analoger Weise der gesamte Triadenspeicher durchgekämmt. Wenn dabei eine freie Seite gefunden wurde, in die alle Permutationen der Triade eingetragen werden konnten, so wird die Nummer dieser Seite in die Entity NAMEA eingetragen und der Seitenzähler von NAMEA erhöht. Ist jedoch keine freie Seite mehr vorhanden, so wird dies in der Page-Tabelle vermerkt, damit bei späterem Aufruf von STRELE zeitraubende Suchvorgänge sich erübrigen. Wie in dem Falle, daß von vornherein festgestellt wurde, daß der Triadenspeicher voll ist, erfolgt jetzt diese Meldung und Rücksprung ins aufrufende Programm. Um die Struktur des Programms klarer darstellen zu können, wird auf der folgenden Seite ein - an einigen Stellen vereinfachtes - Flußdiagramm angegeben.



3.3.4.3 SUBROUTINE DLRELE (NAMEA,NAMEB,NAMEC,MØD)-Delete Relation Entry

Länge : 51

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAMEA(5),NAMEB(5),NAMEC(5),PAGTAB(109)
CALL DLRELE (NAMEA,NAMEB,NAMEC,MØD)

Eingangsparameter : NAMEA : Name "A" der Relation

NAMEB : Name "B" des ersten Entry

NAMEC : Name "C" des zweiten Entry

MØD : Löschmodus

= 0 : nur die Triade wird gelöscht

= 1 : außerdem wird die Entity "C" gelöscht

= 2 : außerdem werden die Entities "C" und "B" gelöscht

= 3 : außerdem werden "C","B" und "A" gelöscht

Ausgangsparameter : -

Transients : -

Externbezüge : RTRNID,ASK,PAGE,ERRØR,DELETE,DELENT

Common : //IP(2048)/IENTI/IE(2048)/DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : Wenn eine der drei Entities "A","B" oder "C" nicht existiert, wird ihr Name mit der Meldung

----- NICHT VORHANDEN -

-----

ausgegeben, und es erfolgt Rücksprung ins aufrufende Programm. Wenn "A" keine Relation-Entity ist, erfolgt die Meldung

***** KEINE RELATION

und ebenfalls Rücksprung

Programmaufbau : Nachdem die Existenz von "A", "B" und "C" sichergestellt wurde und die Klasse von "A" überprüft wurde, wird für jede der Seiten, auf denen Triaden der Relation "A" stehen, DELETE aufgerufen. Falls auf einer Seite eine Triade gelöscht wurde, was dadurch kenntlich gemacht wurde, daß DELETE keinen Reject-Code zurückgibt, wird in der Page-Tabelle vermerkt, daß auf dieser Seite gelöschte Triaden existieren. Je nach dem Wert von MØD wird dann DELENT 0-, 1-, 2- oder 3-mal aufgerufen, um die zu löschen- den Entities zu löschen.

3.3.5 Gruppe G5 (Lesen von Daten)

SUBROUTINE RDENT(NAME, IDATLG, ITYPE, IRC) - Read Entity

Länge : 34

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAME(5)

CALL RDENT(NAME, IDATLG, ITYPE, IRC)

Eingangsparameter : NAME : Name der zu lesenden Entity

IDATLG : Länge (in Worten) des Datenbereichs
der Entity

ITYPE : Klasse der Entity

= 1 : Relation-Entity

= 2 : Single-Value-Entity

= 3 : Multi-Value-Entity

Transients : IRC : Eingang : Nummer des Datenblocks (200 Worte)
der Entity, der zu lesen ist

Ausgang : = 0 : Fehlermeldung; es wurden keine
Daten gelesen

Externbezüge : RTRNID, ERRØR, PAGE

Common : /IENTI/IE(2048)/STACK/ISTACK(200)

Fehlerstops : -

Ausgegebene Texte : Wenn die Entity NAME nicht existiert, erfolgt die Meldung

***** ENTITY EXISTIERT NICHT

und Rücksprung ins aufrufende Programm. Wenn als IRC eine Zahl eingegeben wurde, zu der kein Datenblock existiert, erfolgt die Meldung

***** DATENBEREICH-UEBERSCHREITUNG

und Rücksprung. In beiden Fällen wird IRC = 0 gesetzt.

Programmaufbau : Nachdem die Existenz der Entity sichergestellt wurde, werden zunächst Entity-Klasse, Adresse der Daten und Länge des Datenbereichs bestimmt. Nachdem dann die Gültigkeit der Nummer des Datenblocks überprüft wurde, wird der gewünschte Datenblock linksbündig in das Feld ISTACK geschrieben. Dabei umfaßt der n-te Datenblock die Worte $200(n-1)+1$ bis $200 n$ des Datenbereichs der Entity, so daß jeder Datenblock (mit Ausnahme des letzten) genau 200 Worte umfaßt; der letzte Datenblock kann natürlich kürzer sein. Der im ersten Wort des ersten Datenblocks stehende Synonym-Zähler wird vor dem Übertragen in das Feld ISTACK subtrahiert, so daß er nicht mit ausgegeben wird. Der beim letzten Datenblock nicht beschriebene Teil von ISTACK wird gelöscht d.h. mit Blanks besetzt.

3.3.6 Gruppe G6 (Lesen von Namen)

3.3.6.1 SUBROUTINE RDNAME(NAME,NR,ID1,ID2,ICØD)-Read Name

Länge : 32

wird aufgerufen von : RDSYN,LIELEM,FDRELE,FDRTØP,FDNEXT

Form des Aufrufs : INTEGER NAME(5)

CALL RDNAME(NAME,NR,ID1,ID2,ICØD)

Eingangsparameter : NR : Nummer des Synonyms der Entity, das zu lesen ist

ID1, ID2 : ID der Entity, deren Name zu bestimmen ist

Ausgangsparameter : NAME : Feld, in das der Name der Entity linksbündig übertragen wird

ICØD : Code, der angibt, ob ein sinnvoller Name gelesen wurde :

= 0 : es trat ein Fehler auf; es wurde kein Name gelesen

= 1 : der Name wurde ordnungsgemäß gelesen

= 2 : Sonderfall für FDRELE, wenn ID1 = 0 ist, wird kein Name gelesen aber ICØD = 2 zurückgegeben

Transients : -

Externbezüge : ERRØR, PAGE, ASK

Common : /IENTI/IE(2048)

Fehlerstops : -

Ausgegebene Texte : Wenn ID1 = 0 angegeben wurde, erfolgt die Meldung

***** KEINE ID-INFØRMATION

und Rücksprung ins aufrufende Programm mit ICØD = 2. Wenn zu dem angegebenen ID keine Daten existieren, erfolgt die Meldung

***** ID UNGÜELTIG

und Rücksprung mit ICØD = 0. Wenn NR≤0 oder größer als die Anzahl der Synonyme der Entity angegeben wird, erfolgt die Meldung

***** KEIN SYNØNYM MIT DIESER NUMMER VORHANDEN

und Rücksprung mit ICØD = 0. In allen anderen Fällen wird der Name der Entity mit der Meldung

----- NAME LAUTET

-----

ausgegeben und der Rücksprung erfolgt mit ICØD = 1

Programmaufbau : Nachdem abgefragt wurde, ob ID1≠∅ ist, wird durch Aufruf von PAGE die richtige Seite im Entitätspeicher geholt und geprüft, ob von ID2 ein Zeiger zu einem Datenbereich führt. Schließlich wird nach einer Überprüfung des Wertes von NR der Name der Entity in das Feld NAME linksbündig übertragen. Dabei werden die Synonyme in der Reihenfolge numeriert, in der sie im Datenbereich der Entity stehen, und zwar hat das letzte Synonym, das also unmittelbar am Ende des Datenbereichs steht, die Nummer 1.

3.3.6.2 SUBROUTINE RDSYN (NAME, NAMES, NR, ISYN, ICØD) - Read Synonym

Länge : 19

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAME (5), NAMES (5)

CALL RDSYN (NAME, NAMES, NR, ISYN, ICØD)

Eingangsparameter : NAME : irgendein Name der Entity

NR : Nummer des Synonyms der Entity, das zu lesen ist

Ausgangsparameter : NAMES : Feld, in das der Name des Synonyms linksbündig übertragen wird

ISYN : Anzahl der Synonyme der Entity

ICØD : Code, der angibt, ob ein sinnvoller Name gelesen wurde :

= 0 : es trat ein Fehler auf; kein Name wurde gelesen

= 1 : der Name wurde ordnungsgemäß gelesen

Transients : -

Externbezüge : RTRNID, ERRØR, PAGE, RDNAME

Common : /IENTI/IE(2048)

Fehlerstops : -

Ausgegebene Texte : Wenn die Entity NAME nicht existiert,
erfolgt die Meldung
***** ENTITY EXISTIERT NICHT
und Rücksprung ins aufrufende Programm mit
ICØD = 0 .

Programmaufbau : Nachdem durch Aufruf von RTRNID die Existenz
der Entity NAME sichergestellt und ihr ID
bestimmt wurde, wird durch Aufruf von PAGE
die richtige Seite im Datenspeicher geholt.
Die Adresse der Daten und die Anzahl der Syno-
nyme der Entity werden bestimmt, und durch Auf-
ruf von RDNAME werden Name des Synonyms und
Code bestimmt und ausgegeben. Der Name des
Synonyms steht dann linksbündig im Feld NAMES.

3.3.7 Gruppe G7 (Lesen von Informationen über Mengen)

3.3.7.1 SUBROUTINE FDELEM(NAME,NAMECL,ICØD)-Find Element

Länge : 48

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAME(5),NAMECL(5)
CALL FDELEM(NAME,NAMECL,ICØD)

Eingangsparameter : NAME : Name des Elements
NAMECL : Name der Menge

Ausgangsparameter : ICØD : Code, der angibt, ob NAME ein Element
der Menge ist :
= 0 : keine Elementbeziehung (kann
auch bedeuten, daß eine der bei-
den Entities nicht existiert oder
daß NAMECL keine Menge ist)
= 1 : es besteht eine Elementbeziehung :
NAME ∈ NAMECL

Transients : -

Externbezüge : RTRNID,ERRØR,PAGE,ASK

Common : /IENTI/IE(2048)

Fehlerstops : -

Ausgegebene Texte : Wenn die Entity NAME nicht existiert, erfolgt die Meldung

```

***** ELEMENT EXISTIERT NICHT
und Rücksprung ins aufrufende Programm. Wenn
NAMECL nicht existiert, erfolgt die Meldung
***** MENGE EXISTIERT NICHT
und Rücksprung. Wenn NAMECL keine Multi-Value
Entity ist, erfolgt die Meldung
***** KEINE MENGE
und Rücksprung. Wenn keine Elementbeziehung
zwischen NAME und NAMECL besteht, d.h. wenn
gilt NAME ∉ NAMECL, erfolgt die Meldung
----- KEINE ELEMENTBEZIEHUNG
----- .....
und Rücksprung. In jedem dieser Fälle wird
ICØD=0 zurückgegeben. Wenn dagegen NAME ∈ NAMEC
gilt, wird ICØD=1 gesetzt und es erfolgt die
Meldung
----- ELEMENTBEZIEHUNG ZWISCHEN
----- .....

```

Programmaufbau : Nachdem durch Aufruf von RTRNID die Existenz von NAME und NAMECL sichergestellt wurde und die IDs dieser beiden Entities bestimmt wurden, wird durch Aufruf von PAGE die Seite im Datenspeicher geholt, die die Daten von NAMECL enthält. Nachdem dann überprüft wurde, ob NAMECL eine Multi Value-Entity ist, werden die in den Daten stehenden IDs abgesucht, ob das ID von NAME dabei ist. Falls dieses ID gefunden wurde, wird die Suche abgebrochen. Der entsprechende Text wird durch Aufruf von ASK ausgegeben

3.3.7.2 SUBROUTINE LIELEM(NAME,NAMECL,NR,ID1,ID2)-List Element

Länge : 38

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAME(5),NAMECL(5)

CALL LIELEM(NAME,NAMECL,NR,ID1,ID2)

Eingangsparameter : NAMECL : Name der Menge

Ausgangsparameter : NAME : Feld, in das der Name des Elements
linksbündig übertragen wird

ID1,ID2 : ID des Elements

Transients : NR : Eingang : Nummer des Elements der Menge, das
zu bestimmen ist

Ausgang : = 0 : Fehlermeldung ; ≠ 0 : Anzahl
der Elemente der Menge

Externbezüge : RTRNID,ERRØR,PAGE,ASK,RDNAME

Common : /IENTI/IE(2048)

Fehlerstops : -

Ausgegebene Texte : Wenn NAMECL nicht existiert, erfolgt die
Meldung

***** MENGE EXISTIERT NICHT

und Rücksprung ins aufrufende Programm. Wenn
NAMECL keine Multi-Value-Entity ist, erfolgt
die Meldung

***** KEINE MENGE

und Rücksprung. Wenn $NR \leq 0$ oder größer als die
Anzahl der Elemente der Menge eingegeben wurde,
erfolgt die Meldung

***** NUMMER NICHT IM ELEMENTBEREICH

und Rücksprung. In diesen Fällen wird immer
NR = 0 zurückgegeben.

Programmaufbau : Nachdem durch Aufruf von RTRNID sichergestellt
wurde, daß NAMECL existiert, wird durch Aufruf
von PAGE die richtige Seite im Datenspeicher ge-

holt und überprüft, ob NAMECL eine Multi-Value-Entity ist. Dann wird die Anzahl der Elemente von NAMECL bestimmt und abgecheckt, ob NR eine sinnvolle Elementnummer ist. Dabei werden die Elemente in der Reihenfolge numeriert, in der ihre IDs in der Multi-Value-Entity abgespeichert sind, und zwar hat das Element, das als erstes, also unmittelbar am Anfang des Datenbereichs von NAMECL steht, die Nummer 1. Das ID des Elements mit der Nummer NR wird dann bestimmt, und durch Aufruf von RDNAME wird ein Name ausgegeben und linksbündig in das Feld NAME geschrieben. Dabei wird abgecheckt, ob RDNAME einen Fehlercode (ICØD=0) zurückgibt; in diesem Falle wird NR=0 zurückgegeben, andernfalls wird NR gleich der Anzahl der Elemente von NAMECL gesetzt.

3.3.8 Gruppe G8 (Lesen von Informationen über Relationen)

3.3.8.1 SUBROUTINE FDRELE (NAMEA, NAMEB, NAMEC, IPAGET, IND, ICØD) - Find Relation Entry

Länge : 76

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER NAMEA(5), NAMEB(5), NAMEC(5), PAGTAB(109)
CALL FDRELE (NAMEA, NAMEB, NAMEC, IPAGET, IND, ICØD)

Eingangsparameter : -

Ausgangsparameter : IPAGET : falls die oder eine gesuchte Triade gefunden wurde, die Seite des Triadenspeichers, auf der diese steht, andernfalls unbestimmt

IND : falls die oder eine gesuchte Triade gefunden wurde, die Adresse auf der Seite des Triadenspeichers, auf der sie steht (siehe Abschnitt 2.4.2.3)

ICØD : Code, der angibt, ob eine gültige Triade gefunden wurde :

>0 : eine gültige Triade wurde gefunden
 (siehe Abschnitt 3.4.2.3)
 =0 : es wurde keine Triade gefunden
 =-1 : es wurde eine Triade gefunden,
 aber sie enthält (wenigstens) ein
 ungültiges ID
 =-2 : bei einem Fragetyp, bei dem NAMEA
 bekannt ist, wurde als Entity "A"
 eine Single- oder Multi-Value-Entity
 angegeben
 =-3 : eine der Entities "A", "B" oder "C"
 existiert nicht

Transients : NAMEA : Name "A" der Relation
 NAMEB : Name "B" des ersten Entry
 NAMEC : Name "C" des zweiten Entry
 Namen, nach denen gesucht werden soll, werden 0
 gesetzt (d.h. mit Blanks besetzt). Wurde nur ein
 Name weggelassen, so wird auf diesem Namen, falls
 genau eine Triade gefunden wurde, der Name der ge-
 suchten Entity zurückgegeben. Falls mehrere Triaden,
 d.h. ein C-Ring, gefunden wurden, wird stattdessen
 der Name einer durch PUTRIE erzeugten Multi-Value-
 Entity zurückgegeben, die die Elemente dieses Ringes
 enthält. Wurden zwei Namen weggelassen, so werden
 keine Namen (außer dem einen, der eingegeben wurde)
 zurückgegeben.

Externbezüge : RTRNID,ASK,PAGE,RETRIV,ERRØR,RDNAME

Common : /IENTI/IE(2048)/DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : Wenn eine der angegebenen Entities nicht
 existiert, erfolgt die Meldung
 ----- NICHT VORHANDEN -
 -----
 und Rücksprung ins aufrufende Programm mit
 ICØD=-3. Wenn keine Triade gefunden wurde,

erfolgt die Meldung

***** TRIADE NICHT VORHANDEN

und Rücksprung mit $IC\emptyset D=0$. Wenn eine Triade gefunden wurde, die aber ein ID enthält, das ungültig ist, d.h. zu dem keine Daten existieren, erfolgt die Meldung

***** TRIADE UNGÜELTIG

und Rücksprung mit $IC\emptyset D=-1$. Wenn eine gültige Triade gefunden wurde, erfolgt die Meldung

***** TRIADE GEFUNDEN

und Rücksprung mit $IC\emptyset D>0$. Wenn NAMEA angegeben wurde, aber keine Relation-Entity ist, erfolgt die Meldung

***** KEINE RELATION

und Rücksprung mit $IC\emptyset D=-2$.

Programmaufbau : Durch Aufruf von RTRNID wird festgestellt, ob angegebene Entities NAMEC, NAMEB oder NAMEA existieren, und ihre IDs werden bestimmt. Die IDs nicht angegebener Namen werden 0 gesetzt. Je nachdem, ob NAMEA angegeben wurde oder nicht, läuft der nächste Bearbeitungsschritt anders ab: Falls NAMEA nicht angegeben wurde, also keine Relation angegeben ist, muß für jede Seite des Triadenspeichers RETRIV aufgerufen werden. Andernfalls wird abgecheckt, ob NAMEA eine Relation-Entity ist, und RETRIV wird nur für die Seiten des Triadenspeichers aufgerufen, auf denen diese Relation steht. Dieses Vorgehen ist ausreichend, da ja alle Permutationen einer Triade immer auf der gleichen Seite stehen. In jedem Falle wird die Suche abgebrochen, wenn eine Triade gefunden wurde, d.h. wenn RETRIV mit positivem Code zurückkommt. RETRIV gibt, wenn nur ein ID null gesetzt war, auf diesen Parameterpositionen ein ID als Ergebnis zurück. Wurde nur eine Triade, also im B-Ring ein Terminal, gefunden, dann ist das zurückgegebene ID das der C-Entity des Terminals. (Durch die Verwen-

dung verschiedener Permutationen einer Triade kann sich das Retrieval ja auf Fragen der Typen ABC, AB? und A?? beschränken / 6 /, / 11 /.) Wurde stattdessen ein C-Ring gefunden, so ruft RETRIV die Routine PUTRIE auf, die die Elemente dieses Ringes (bei mehr als 100 nur die vom Top an gezählten ersten 100) in eine automatisch erzeugte Multi-Value-Entity packt, und deren ID wird dann zurückgegeben. Wenn zwei IDs null gesetzt waren, liegt intern der Fragetyp A?? vor, und es werden von RETRIV keine IDs zurückgegeben. Stattdessen wird bei erfolgreicher Suche die Adresse des Top des gefundenen B-Ringes zurückgegeben. Diese Adresse steht dann in den Variablen IPAGET und IND. Wenn RETRIV mit positivem Code zurückkam, werden die Namen, die zu den von RETRIV übergebenen IDs gehören, durch Aufruf von RDNAME ausgegeben und in die entsprechenden Felder übertragen. Dabei wird noch die Gültigkeit der IDs und damit der Triade überprüft.

Beispiele : Weggelassene Namen, d.h. Namen, deren Felder 0 gesetzt sind, werden in diesen Beispielen durch "┐" angegeben. Wir gehen von einer Menge aus, die aus den Triaden

ABC und ABD

in allen Permutationen (d.h. Permutationstyp 2) besteht. Ferner existiere noch eine Single-Value-Entity E. Die Frage

A ┐ C führt zur Antwort ABC - ***** TRIADE GEFUNDEN
 Die Frage

A,B ┐ führt zur Antwort A B*OC - ***** TRIADE GEFUNDEN

Dabei ist *OC eine Multi-Value-Entity mit den Elementen C und D. Die Frage

ABD führt zur Antwort ABD - ***** TRIADE GEFUNDEN
 Die Frage

ABE oder AE..o.ä. führt zur Antwort ***** TRIADE NICHT
 VORHANDEN

Die Frage

EBE oder E $\sqcup$ $\sqcup$ o.ä. führt zur Antwort

***** KEINE RELATION

Die Frage

A $\sqcup$ $\sqcup$ führt zur Antwort ***** TRIADE GEFUNDEN

Die Variablen IPAGET und IND enthalten dann die Adresse von B (des Top des B-Ringes) im Triadenspeicher.

3.3.8.2 SUBROUTINE FDRTOP (NAME,IPAGET,IND,ICOD)-Find Ring-Top

Länge : 22

wird aufgerufen von : DIALOG

Form des Aufrufs : INTEGER NAME(5),PAGTAB(109)

CALL FDRTOP (NAME,IPAGET,IND,ICOD)

Eingangsparameter : IPAGET : Seite im Triadenspeicher, auf der
das gesuchte Element steht

IND : Adresse des Elements auf dieser Seite

Ausgangsparameter : NAME : Feld, in das der Name des Elements
linksbündig übertragen wird

ICOD : Code, der angibt, ob ein sinnvoller Name
gelesen wurde :

= 0 : es trat ein Fehler auf; kein Name
wurde gelesen

= 1 : der Name wurde ordnungsgemäß gele-
sen

= 2 : an der angegebenen Stelle stand
kein ID

Transients : -

Externbezüge : ERROR,PAGE,RDNAME

Common : //IP(2048)/DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : Wenn IPAGET>PAGTAB(8) oder IPAGET<PAGTAB(7)
angegeben wird, erfolgt die Meldung

***** FALSCH SEITENNUMMER

und Rücksprung ins aufrufende Programm mit $IC\emptyset D=0$. Wenn eine der folgenden Bedingungen verletzt wird :

$IP(2047) < IND \leq 2046$

$(IND \leq IP(2048) \wedge IND \equiv 0 \bmod 3) \vee (IND > IP(2048) \wedge IND \equiv 1 \bmod 3)$

erfolgt die Meldung

***** FALSCHER INDEX

und Rücksprung mit $IC\emptyset D=0$. (IP(2047) enthält die Adresse der letzten Zelle von F1, IP(2048) die der letzten Zelle von HAS; siehe dazu Abschnitt 2.2.2.2)

Programmaufbau : Nachdem die Gültigkeit von Seitennummer und Index überprüft wurde und dabei die richtige Seite des Triadenspeichers geholt wurde, wird das gesuchte ID gelesen. Durch Aufruf von RDNAME wird der Name der Entity ausgegeben und in das Feld NAME übertragen. Dabei wird auch die Gültigkeit des ID überprüft. Das Programm kann mit B- und C-Zellen arbeiten.

3.3.8.3 SUBROUTINE FDNEXT(NAME,IPAGET,IND,IC $\emptyset$ D)-Find Next

Länge : 29

wird aufgerufen von : DIAL $\emptyset$ G

Form des Aufrufs : INTEGER NAME(5), PAGTAB(109)

CALL FDNEXT(NAME,IPAGET,IND,IC $\emptyset$ D)

Eingangsparameter : IPAGET : Seite im Triadenspeicher, auf der das gesuchte Element steht

Ausgangsparameter : NAME : Feld, in das der Name des Elements linksbündig übertragen wird

IC $\emptyset$ D : Code, der angibt, ob ein sinnvoller Name gelesen wurde

= 0 : es trat ein Fehler auf; kein Name wurde gelesen

= 1 : der Name wurde ordnungsgemäß gelesen

= 2 : an der angegebenen Stelle stand kein ID

Transients : IND : Eingang : Adresse des bekannten Ringelements
 Ausgang : falls das angegebene Element Bottom
 war, unverändert;
 sonst : Adresse des nächsten Elements
 auf dem Ring

Externbezüge : ERRØR,PAGE,RDNAME

Common : //IP(2048)/DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : Wenn IPAGET falsch angegeben wurde (siehe
 3.3.8.2) erfolgt die Meldung
 ***** FALSCHER SEITENNUMMER
 und Rücksprung ins aufrufende Programm mit
 ICØD=0. Wenn IND falsch angegeben wurde (siehe
 ebenfalls 3.3.8.2),erfolgt die Meldung
 ***** FALSCHER INDEX
 und Rücksprung mit ICØD=0. Wenn schließlich
 das angegebene Element Bottom des Ringes war,
 erfolgt die Meldung
 ***** BØTTØM SCHON ERREICHT
 und ebenfalls Rücksprung.

Programmaufbau : Nachdem die Gültigkeit der Seitennummer sicher-
 gestellt wurde, wird durch Aufruf von PAGE die
 richtige Seite im Triadenspeicher geholt und die
 Gültigkeit des Index überprüft. Dann wird fest-
 gestellt, ob es sich um eine Bottom-Zelle handelt
 falls nicht, wird der Index durch den Index der
 nächsten Zelle auf dem Ring ersetzt und das ge-
 suchte ID gelesen. Durch Aufruf von RDNAME wird
 der Name der Entity ausgegeben und in das Feld
 NAME übertragen. Dabei wird auch die Gültigkeit
 des ID überprüft. Das Programm kann auf B- und
 C-Ringen, aber nicht auf Conflict-Ringen arbeiten

3.3.9 Gruppe G9 (Systemprogramme)

3.3.9.1 SUBROUTINE SYSINT(INL,IPL,IEL,N1,NH,NC,LBUFF)-System
Initialisierung

Länge : 50

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER PAGTAB(109)

CALL SYSINT(INL,IPL,IEL,N1,NH,NC,LBUFF)

Eingangsparameter : INL : gewünschte Seitenzahl für den Namenspeicher
 IPL : gewünschte Seitenzahl für den Triadenspeicher
 IEL : gewünschte Seitenzahl für den Entityspeicher
 LBUFF : Anzahl der Character, die von einer
 Eingabezeile zu interpretieren sind

Ausgangsparameter : -

Transients : N1 : Eingang: Anzahl der A-Zellen in F1
 Ausgang: Adresse des letzten Wortes von F1
 NH : Eingang: Anzahl der B-Zellen in HAS
 Ausgang: Adresse des letzten Wortes von HAS
 NC : Eingang: Anzahl der Zellen in der Hash-Area einer
 Seite des Namenspeichers
 Ausgang: Adresse des ersten Wortes der Conflict-
 Area einer Seite des Namenspeichers

Externbezüge : ERRØR,PAGE,NSINIT,F1INIT,HSINIT,F2INIT,DSINIT

Common ://IP(2048)/INAME/IN(2048)/IENTI/IE(2048)/DT/PAGTAB/BUFFER/
 IBUF(80),LBUF

Fehlerstops : Wenn LBUFF>80 angegeben wird, erfolgt STØP mit der
 Meldung
 ***** BUFFERLAENGE UEBERSCHRITTEN
 Wenn LBUFF<30 angegeben wird, erfolgt STØP mit der
 Meldung
 ***** BUFFERLAENGE NICHT AUSREICHEND
 Wenn INL+IPL+IEL>100 angegeben wird, erfolgt STØP
 mit der Meldung
 ***** SYSTEMKAPAZITAET UEBERSCHRITTEN
 Wenn $N1+6*NH \geq 2048$ oder $8*NC \geq 2047$ angegeben wird,
 erfolgt STØP mit der Meldung

***** SEITENLAENGE UEBERSCHRITTEN

Ausgegebene Texte : -

Programmaufbau : Nachdem überprüft wurde, ob die angegebenen Systemparameter sinnvoll sind, wird für jede der Seiten von Namen-, Triaden- und Datenspeicher PAGE aufgerufen und die Struktur der Seite dann durch NSINIT bzw. F1INIT, HSINIT und F2INIT bzw. DSINIT initialisiert. Dabei wird außerdem PAGTAB mit den Startwerten geladen :

PAGTAB(1) = 1	PAGTAB(I) = 1 für $10 \leq I \leq INL+9$
PAGTAB(2) = INL+1	PAGTAB(I) = 2 für $10+INL \leq I \leq INL+IPL+9$
PAGTAB(3) = INL+IPL+1	PAGTAB(I) = 259 für $10+INL+IPL \leq I \leq INL+I$
$PAGTAB(4) = \begin{cases} 2 & \text{für } INL > 1 \\ 0 & \text{für } INL = 1 \end{cases}$	PAGTAB(7) = INL
$PAGTAB(5) = \begin{cases} INL+2 & \text{für } IPL > 1 \\ 0 & \text{für } IPL = 1 \end{cases}$	PAGTAB(8) = INL+IPL
$PAGTAB(6) = \begin{cases} INL+IPL+2 & \text{für } IEL > 1 \\ 0 & \text{für } IEL = 1 \end{cases}$	PAGTAB(9) = INL+IPL+IEL

3.3.9.2 SUBROUTINE PAGE(N, ID)-Seitentransport

Länge : 30

wird aufgerufen von : STELEM, DLELEM, DLSET, STRELE, DLRELE, RDENT, RDSYN, FDELEM, LIELEM, FDRELE, FDRTØP, FDNEXT, SYSINT, SYSØUT, SYSGCL, RDNAME, EXDATA, PUTRIE, STENT, DELENT, RTRNID, HASHNM

Form des Aufrufs : INTEGER PAGTAB(109)
CALL PAGE(N, ID)

Eingangsparameter : N : gibt an, welche Art von Seite transportiert werden soll
 = 1 : eine Seite des Namensspeichers
 = 2 : eine Seite des Triadenspeichers
 = 3 : eine Seite des Datenspeichers
 ID : Nummer der zu holenden Seite

= 0 : es ist keine Seite zu holen; nur
die aktuelle Seite ist auf Platte
zu schreiben

Ausgangsparameter : -

Transients : -

Externbezüge : ERRØR

Common : //IP(2048)/INAME/IN(2048)/IENTI/IE(2048)/DT/PAGTAB

Fehlerstops : Wenn $N \leq 0$ oder $N > 3$ angegeben wird, erfolgt STØP
mit der Meldung

***** FALSCHER SEITENTYP

Wenn eine der folgenden Bedingungen verletzt wird :

$N=1 \Rightarrow 1 \leq ID \leq PAGTAB(7)$

$N=2 \Rightarrow PAGTAB(7)+1 \leq ID \leq PAGTAB(8)$

$N=3 \Rightarrow PAGTAB(8)+1 \leq ID \leq PAGTAB(9)$

erfolgt STØP mit der Meldung

***** FALSCHER SEITENNUMMER

Ausgegebene Texte : -

Programmaufbau : Nachdem die Richtigkeit der Eingangsparameter
überprüft wurde, wird festgestellt, ob sich die
gewünschte Seite schon im Kernspeicher befindet;
in diesem Fall erfolgt Rücksprung ins aufrufende
Programm. Andernfalls wird die im Kernspeicher
befindliche Seite auf den ihr zugewiesenen Platz
auf einem externen Speichermedium mit der logischen
Gerätenummer 10 geschrieben und von dort die
Seite mit der Nummer $ID (\neq 0)$ in den Kernspeicher
gelesen. Außerdem wird $PAGTAB(N)=ID$ gesetzt.
Diese beiden letzten Schritte entfallen, wenn
 $ID=0$ angegeben wurde.

3.3.9.3 SUBRØUTINE SYSGCL - System Garbage Collection

Länge : 14

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER PAGTAB(109)

CALL SYSGCL

Parameter : -

Externbezüge : PAGE, GARCØL

Common : /DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Für jede Seite des Triadenspeichers wird anhand der Page-Tabelle festgestellt, ob sie gelöschte Triaden enthält, und in diesem Fall wird die Seite mit PAGE in den Kernspeicher geholt und mit GARCØL bereinigt.

3.3.9.4 SUBROUTINE SYSØUT - System out

Länge : 9

wird aufgerufen von : DIALØG

Form des Aufrufs : INTEGER PAGTAB(109)

CALL SYSØUT

Parameter : -

Externbezüge : PAGE

Common : /DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Durch Aufruf von PAGE mit 0 als zweiten Parameter werden die aktuellen, im Kernspeicher befindlichen Seiten auf Platte geschrieben. Anschließend wird der aktuelle Stand der Page-Tabelle ebenfalls auf die Platte übertragen, die mit der logischen Gerätenummer 10 angesprochen wird.

3.4 Die Systemdienste der Ebene 2

3.4.1. Gruppe S1 (Manipulation von Entities)

3.4.1.1 SUBROUTINE STENT(NAME, IDATLG, IDATA, ID1, ID2, ITYPE) - Store Entity

Länge : 141

wird aufgerufen von : STSNGL, CHSNGL, CHNAME, DECSYN, DECSET,
STELM, DECREL, STRELE, PUTRIE, DIALØG

Form des Aufrufs : INTEGER NAME(5), IDATA(IDATLG), PAGTAB(109)
CALL STENT(NAME, IDATLG, IDATA, ID1, ID2, ITYPE)

Eingangsparameter : NAME : Name der Entity, max. 10 Characters,
die linksbündig stehen müssen. Ein
Name, der mit zwei Blanks anfängt,
ist nicht erlaubt.

IDATLG : Länge des Datenblocks (in Worten)

IDATA : Feld, das die Daten enthält

ID1, ID2 : ID der Entity, zu der NAME synonym
gesetzt werden soll

= 0 : NAME ist als eigene Entity
abzuspeichern

ITYPE : Typ (Klasse) der abzuspeichernden
Entity :

= 1 : Relation-Entity

= 2 : Single-Value-Entity

= 3 : Multi-Value-Entity

IDATLG, IDATA und ITYPE werden ignoriert, wenn
ID1 ≠ 0 angegeben wird.

Ausgangsparameter : -

Transients : -

Externbezüge : ERRØR, RTRNID, PAGE, ALCNAM, ASK, DELENT, DSHIFT

Common : /DT/PAGTAB/IENTI/IE(2048)

Fehlerstops : -

Ausgegebene Texte : Wenn NAME(1)=0 ist, d.h. wenn der abzuspeichernde Name mit zwei Blanks beginnt, erfolgt die Meldung

***** ILLEGALER NAME

und Rücksprung ins aufrufende Programm.

Wenn auf keiner Seite des Datenspeichers ein ID frei ist, erfolgt die Meldung

***** DATENSPEICHER VÖLL

und Rücksprung. Wenn auf keiner Seite des Datenspeichers genügend Platz ist, um den Datenbereich von NAME aufzunehmen, erfolgt die Meldung

***** DATENBLÖCK ZU LANG FÜR FREIEN
SPEICHER

und Rücksprung. Wenn die Entity NAME schon existiert, so wird der Anwender gefragt

----- ÜBERSPEICHERN VON

-----

Sind die beiden ersten Buchstaben der Antwort des Anwenders JA oder ØK, so wird die alte Entity NAME gelöscht und NAME wie gewünscht abgespeichert, andernfalls erfolgt ein Rücksprung ins aufrufende Programm.

Sind bei einer Synonym-Abspeicherung weniger als 5 Worte auf der Seite des Datenspeichers frei, so erfolgt die Meldung

***** ÜBERLAUF BEI SYNONYM-SPEICHERUNG

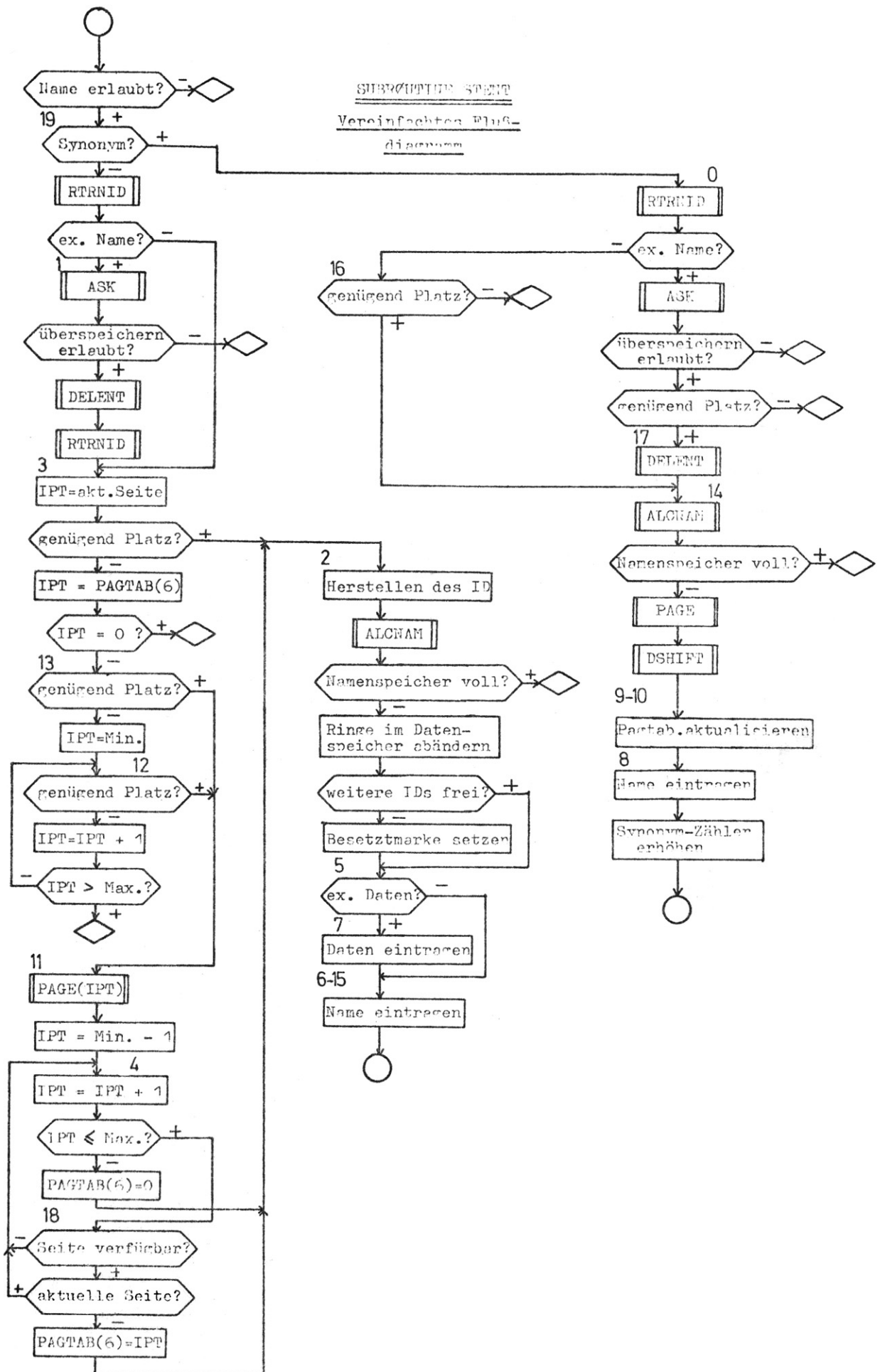
und Rücksprung. In diesem, wie auch den ersten drei genannten Fällen unterbleibt eine Abspeicherung.

Programmaufbau : Nachdem abgefragt wurde, ob NAME ein erlaubter Name ist, d.h. ob nicht die beiden ersten Characters von NAME Blank sind, wird festgestellt, ob ID1=0 ist. Je nachdem, ob das der Fall ist oder nicht, werden verschiedene Programmteile durchlaufen. Wenn ID1=0 ist, wird NAME als eigene Entity abgespeichert, im anderen Falle wird NAME

als neues Synonym zu der Entity, deren ID mit den Parametern ID1 und ID2 angegeben wurde, abgespeichert. Im ersten Falle wird durch Aufruf von RTRNID festgestellt, ob schon eine Entity mit dem Namen NAME existiert. Falls ja, wird mit ASK gefragt, ob diese Entity gelöscht werden soll. Wenn dazu keine Erlaubnis gegeben wird, erfolgt Rücksprung; andernfalls wird durch Aufruf von DELENT die alte Entity NAME gelöscht und durch Aufruf von RTRNID festgestellt, an welcher Stelle des Namenspeichers der Name NAME einzutragen ist. Die folgenden Schritte laufen in jedem Falle ab, gleichgültig, ob eine Entity NAME schon existiert hat oder nicht. Wenn auf der gerade im Kernspeicher befindlichen Seite des Datenspeichers noch Platz ist, kann sofort mit dem Abspeichern der Entity begonnen werden; andernfalls wird zuerst eine freie Seite im Datenspeicher gesucht. Ist keine Seite mehr frei, so erfolgt Rücksprung. Wurde eine freie Seite gefunden, so wird sie in den Kernspeicher geholt. Dann wird die Seitentabelle abgesucht, ob noch freie Seiten vorhanden sind, und das Ergebnis dieser Suche - die Nummer einer freien Seite oder, falls keine gefunden wurde, eine Null - wird in PAGTAB(6) geschrieben. Das Abspeichern der Entity beginnt mit dem Herstellen ihres ID, und zwar wird ID1=PAGTAB(3) und ID2 gleich dem Index des Top des vacant-Ringes gesetzt. Diese Vorgehensweise, zusammen mit dem bei DELENT angewendeten Verfahren, das freige-wordene ID vorne, also als neues Top, in den vacant-Ring einzufügen, gewährleistet, daß sich bei Aufruf von CHSNGL das ID nicht ändert, vorausgesetzt, die neuen Daten passen auf dieselbe Seite wie die alten. Name und ID werden dann durch Aufruf von ALCNAM im Namenspeicher abgespeichert, wenn dieser nicht voll ist; in diesem Falle erfolgt Rücksprung mit einer von ALCNAM erzeugten Meldung. Als nächstes werden die Ringe im Datenspeicher aktu-

alisiert, und zwar wird das neue ID vom Top des vacant-Ringes zum Top des occupied-Ringes. Falls dann der Top-Zeiger des vacant-Ringes 0 ist, also der vacant-Ring leer ist, wird die Seite als besetzt markiert. Anschließend werden die Daten und der Name in den Datenbereich der Seite eingetragen, und zwar werden sie an die vorhandenen Daten hinten angehängt. Dabei wird gleichzeitig das Startregister IE(258) aktualisiert und der Zeiger von der ID-Zelle auf den Datenbereich der Entity eingerichtet.

Wenn ID1 $\neq$ 0 angegeben wurde, wird STENT zur Abspeicherung eines Synonyms verwendet. Durch RTRNID wird dann festgestellt, ob schon eine Entity NAME existiert und, diese wird dann gegebenenfalls nach Aufruf von ASK durch Aufruf von DELENT gelöscht. Wenn auf der Seite der Entity, die das neue Synonym erhalten soll, nicht mehr genug Platz für die Eintragung eines Namens ist, erfolgt in jedem Falle zuerst Rücksprung (also noch vor einem etwaigen Aufruf von DELENT). Nachdem dann durch ALCNAM Name und ID im Namenspeicher abgespeichert wurden (bzw. ein Rücksprung erfolgte, weil dieser voll ist), wird durch Aufruf von PAGE die richtige Seite im Datenspeicher geholt und auf ihr durch Aufruf von DSHIFT genügend Platz für die Eintragung des Namens reserviert. Schließlich wird die Page-Tabelle aktualisiert, der Name eingetragen und der Synonym-Zähler der Entity erhöht. Die Entity, deren ID mit den Parametern ID1 und ID2 angegeben wurde, ist jetzt zusätzlich unter dem Namen NAME ansprechbar. Um die Struktur des Programms zu verdeutlichen, wird auf der folgenden Seite ein - an einigen Stellen vereinfachtes - Flußdiagramm angegeben.



3.4.1.2 SUBROUTINE DELENT (NAME, ID1, ID2, ITYPE, MØD) - Delete Entity

Länge : 131

wird aufgerufen von : CHSNGL, DLSNGL, CHNAME, DECSYN, DLNAME,
DLELEM, DLSET, DLRELE, STENT, DIALØG

Form des Aufrufs : INTEGER NAME(5), PAGTAB(109)
CALL DELENT (NAME, ID1, ID2, ITYPE, MØD)

Eingangsparameter : NAME : zu löschender Name bzw. Name der zu
löschenden Entity

MØD : Löschmodus :

=0 : nur der Name wird gelöscht

=1 : die ganze Entity wird gelöscht

Ausgangsparameter : ID1, ID2 : ID der Entity NAME
=0, wenn diese Entity nicht existiert

ITYPE : Klasse der Entity NAME :

= 1 : Relation-Entity

= 2 : Single-Value-Entity

= 3 : Multi-Value-Entity

Transients : -

Externbezüge : RTRNID, PAGE, ERRØR, HASHNM, DSHIFT

Common : /DT/PAGTAB/INAME/IN(2048)/IENTI/IE(2048)

Fehlerstops : Wenn im Namenspeicher die Bottom-Zelle einer
Conflict-Kette gelöscht werden soll, die aber den
Namen NAME nicht enthält, erfolgt STØP mit Dump
der aktuellen Seite des Namenspeichers und der
Meldung

***** FEHLER BEI NAME-CØNFLICT

Wenn im Datenspeicher ein nicht vorhandenes Synonym
gelöscht werden soll, das aber im Namenspeicher vor-
handen war, erfolgt STØP mit Dump der aktuellen Seite
des Datenspeichers und der Meldung

***** FEHLER BEI SYNØNYM-LØESCHUNG

Diese beiden Fehler konnten bis jetzt noch durch
keine noch so falsche Eingabe provoziert werden.

Ausgegebene Texte : Wenn DELENT mit MØD = 0 für eine Entity aufge-
rufen wird, die nur einen einzigen Namen hat,

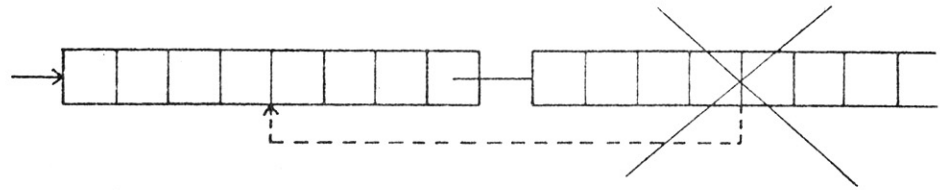
erfolgt die Meldung

***** ILLEGALER LØESCHVERSUCH

und Rücksprung ins aufrufende Programm,
ohne daß der Name gelöscht wurde.

Programmaufbau : Je nachdem, ob MØD=0 oder 1 ist, werden teilweise verschiedene Programmteile durchlaufen. Bei MØD=0 wird zuerst der Name NAME im Kernspeicher gelöscht; dann werden die Daten der Entity im Datenspeicher nach dem Namen NAME abgesucht und dieser Name gelöscht. Bei MØD=1 werden zuerst die Namen der Entity der Reihe nach, beginnend mit dem, der als letzter im Datenbereich steht, im Namenspeicher gelöscht; dann schließlich werden die Daten und Namen auch im Datenspeicher gelöscht. Im Einzelnen ergibt sich dabei folgender Programmaufbau :

Zunächst wird durch Aufruf von RTRNID festgestellt, ob die Entity NAME existiert, falls nicht, erfolgt ohne Kommentar ein Rücksprung ins aufrufende Programm. Durch Aufruf von PAGE wird dann die richtige Seite im Datenspeicher geholt, und Adresse der Daten, Klasse der Entity und Anzahl ihrer Synonyme werden bestimmt. Ist letztere = 1 und gleichzeitig MØD=0, so erfolgt Rücksprung ins aufrufende Programm. Bei MØD=1 wird nun das im Datenspeicher an letzter Stelle stehende Synonym von NAME bestimmt und im Namenspeicher gelöscht, bei MØD=0 wird direkt NAME im Namenspeicher gelöscht. Dazu wird zuerst durch Aufruf von RTRNID die richtige Seite im Namenspeicher geholt und die Adresse des Namens auf dieser Seite bestimmt. Dann wird festgestellt, ob der Name Bottom einer Conflict-Kette ist; falls nicht, wird der nächste Name in dieser Kette bestimmt und mit seinem ID in die zu löschende Zelle übertragen, und die Zelle dieses anderen Namens wird gelöscht :



Wenn der Name Bottom einer Conflict-Kette ist, wird festgestellt, ob die Kette mehr als ein Element besitzt, falls nicht, kann die Zelle sofort gelöscht werden, andernfalls wird der Kopf der Kette bestimmt und von diesem ausgehend die Kette nach dem Vorgänger des zu löschenden Namens, also der zweituntersten Zelle der Kette, abgesucht. Deren Conflict-Information wird gelöscht, also die letzte Zelle der Kette einfach "abgeschnitten". Das Löschen der Zelle besteht in einem einfachen Nullsetzen aller acht Worte, wenn die Zelle in der Hash-Area liegt; andernfalls werden nur die ersten sieben Worte gelöscht und die Zelle durch Einschreiben der Startadresse der vacant-Kette in das achte Wort und Abändern des Startregisters IN(2048) am Anfang der vacant-Kette eingefügt. Bei $M\emptyset D=0$ wird jetzt der Synonymzähler um 1 erniedrigt, der Name NAME im Datenspeicher gesucht und durch Verschieben der hintendranstehenden Namen der Entity und anschließenden Aufruf von DSHIFT gelöscht. Bei $M\emptyset D=1$ werden stattdessen nach dem beschriebenen Verfahren alle Namen der Entity bestimmt und im Namenspeicher gelöscht. Anschließend werden Namen und Daten im Datenspeicher durch Aufruf von DSHIFT gelöscht. Das freigewordene ID wird aus dem occupied-Ring herausgelöst und als neues Top in den vacant-Ring eingefügt (siehe 3.4.1.1). Vor dem Rücksprung ins aufrufende Programm wird bei $M\emptyset D=0$ und bei $M\emptyset D=1$ noch die Page-Tabelle aktualisiert.

3.4.1.3 SUBROUTINE DSHIFT(ISTART,ISTEP) - Data Shift

Länge : 73

wird aufgerufen von : STELEM,DLELEM,DLSET,STRELE,STENT,DELENT

Form des Aufrufs : CALL DSHIFT(ISTART,ISTEP)

Eingangsparameter : ISTART : erstes Wort des Datenbereichs,
dessen Länge verändert wird

ISTEP : Anzahl der Worte, um die die Länge
dieses Bereiches verändert wird :
>0 : der Bereich wird verlängert
<0 : der Bereich wird verkürzt

Ausgangsparameter : -

Transients : -

Externbezüge : ERRØR

Common : /IENTI/IE(2048)

Fehlerstops : Wenn bei ISTEP>0 alle Datenzeiger des occupied-Ringes abgeändert wurden, ohne daß im Datenbereich ISTART erreicht oder unterschritten wurde, erfolgt Fehlerstop mit Dump der aktuellen Seite des Datenspeichers und der Meldung

***** FEHLER IM ALGORITHMUS

Dieser Fehler konnte bisher noch durch keine noch so falsche Eingabe provoziert werden.

Ausgegebene Texte : -

Programmaufbau : Das Programm verschiebt alle Datenbereiche, die bei einer Adresse > ISTART beginnen, um ISTEP Worte nach hinten, wenn ISTEP>0 gilt, und um -ISTEP Worte nach vorne, wenn ISTEP<0 gilt. Dabei wird der bei ISTART beginnende Datenbereich um ISTEP Worte verlängert bzw. verkürzt, wobei seine Daten aber nicht verschoben werden. Alle next-data-Zeiger (siehe Abschnitt 2.2.2.3) von ISTART an werden entsprechend modifiziert, ebenso die Datenzeiger des occupied-Ringes. Dabei macht das Programm von der Tatsache Gebrauch, daß die Datenbereiche in derselben Reihenfolge ange-

ordnet sind wie die zugehörigen ID-Zellen auf dem occupied-Ring, wobei der erste Datenbereich zum Bottom und der letzte zum Top des occupied-Ringes gehört, da durch STENT immer neue IDs als neues Top des occupied-Ringes eingefügt und die zugehörigen Daten hinten an die schon vorhandenen angehängt werden. Die einzelnen Bearbeitungsschritte laufen je nach dem Vorzeichen von ISTEP in umgekehrter Reihenfolge ab, so daß DSHIFT in zwei etwa gleichgroße Hälften zerfällt, von denen jeweils nur eine durchlaufen wird. Bei ISTEP>0 werden die einzelnen Datenbereiche vom occupied-Ring aus abgearbeitet, vom Top abwärts, also von hinten nach vorn fortschreitend. Dagegen werden die Bereiche bei ISTEP<0 von ISTART ausgehend von vorn nach hinten abgearbeitet und erst anschließend die Datenzeiger des occupied-Ringes, wieder vom Top abwärts, abgeändert. In beiden Fällen wird außerdem das Startregister IE(258) abgeändert.

Warnung : Wenn ISTART nicht die Adresse der ersten Zelle eines Datenbereichs ist, oder wenn ISTEP<ISTART-IE(ISTART) gewählt wird, kann die Routine die Struktur der Seite zerstören! Diese Fälle werden nicht abgecheckt, da DSHIFT nicht für den direkten Gebrauch des Anwenders bestimmt ist, sondern nur als Unterprogramm anderer Programme eingesetzt wird.

3.4.1.4 SUBROUTINE RTRNID(NAME,IPAGE,IADR,ID1,ID2,IØC) - Return ID

Länge : 34

wird aufgerufen von : CHNAME,DECSYN,STELM,DLELEM,DLSET,STRELE,
DLRELE,RDENT,RDSYN,FDELEM,LIELEM,FDRELE,
PUTRIE,STENT,DELENT

Form des Aufrufs : INTEGER NAME(5)

CALL RTRNID(NAME,IPAGE,IADR,ID1,ID2,IØC)

Eingangsparameter : NAME : Name der Entity, deren ID zu bestimmen ist

Ausgangsparameter : IPAGE : Zum Namen gehörende Seitennummer im Namenspeicher

IADR : zum Namen gehörende Adresse auf dieser Seite, und zwar :
wenn die Zelle leer ist, die Adresse der Zelle, in die der Name abzuspeichern ist, andernfalls die Adresse der Endzelle einer Conflict-Kette, an die die zum Namen gehörende Zelle anzuhängen ist, bzw. die Adresse des Namens

ID1, ID2 : ID der Entity Name,
= 0, wenn diese Entity nicht existiert

IØC : occupied-Code, gibt den Status der Zelle an :

= 0 : Zelle im Namenspeicher ist leer oder enthält den angegebenen Namen

= 1 : Zelle im Namenspeicher enthält einen anderen als den angegebenen Namen

Transients : -

Externbezüge : HASHNM, PAGE

Common: /INAME/IN(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Durch Aufruf von HASHNM wird die zum Namen NAME gehörende Hash-Adresse bestimmt, die aus der Seitennummer IPAGE im Namenspeicher und der Adresse auf dieser Seite besteht. Da zur Zeit noch keine Überlauf-Seiten für den Namenspeicher vorhesehen sind, ist IPAGE schon die Seite, auf der der Name steht bzw. auf die er zu schreiben ist. Diese Seite wird also durch Aufruf von PAGE in den Namenspeicher geholt, und es wird abgefragt, ob die durch die Hash-Adresse angegebene Zelle leer ist. In diesem Falle wird als ID und als IØC null zurückgegeben. Wenn die Zelle besetzt ist,

kann sie als Anfang einer (eventuell nur aus dieser einen Zelle bestehenden) Conflict-Kette aufgefaßt werden. Diese Kette wird dann abgesehen, ob sie den Namen NAME enthält. Wenn ja, wird das zum Namen gehörende ID und IØC = 0 zurückgegeben, andernfalls IØC=1 und 0 als ID.

3.4.2 Gruppe S2 (Manipulation von Triaden)

3.4.2.1 SUBROUTINE STØRE(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2,IRC) - Store Triade

Länge : 149

wird aufgerufen von : STRELE

Form des Aufrufs : CALL STØRE(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2,IRC)

Eingangsparameter : IDA1,IDA2 : ID der Relation "A"

IDB1,IDB2 : ID des ersten Entry "B"

IDC1,IDC2 : ID des zweiten Entry "C"

Ausgangsparameter : IRC : Reject-Code :

= 0 : alle Permutationen wurden ordnungsgemäß abgespeichert

= 1 : wenigstens eine Permutation konnte nicht abgespeichert werden

Transients : -

Externbezüge : EXDATA,ERRØR,ALØØK,AALLØC,IHASH,BRETRI,CRETRI,
KEEPFR,GØBØTM

Common : //IP(2048)

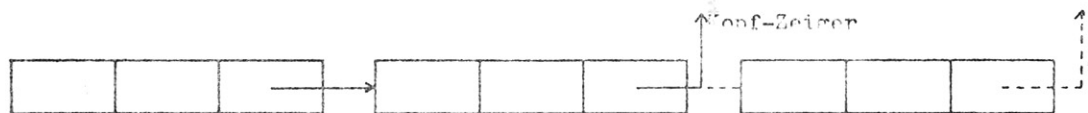
Fehlerstops : -

Ausgegebene Texte : Wenn die Entity "A" keine Relation-Entity ist, erfolgt die Meldung
***** FALSCHER ENTITY-CLASS
und Rücksprung ins aufrufende Programm mit
IRC=1 .

Programmaufbau : Zunächst wird durch Aufruf von EXDATA sichergestellt, daß die Entity "A" eine Relation-Entity ist. Weiterhin wird der Permutationstyp von "A" bestimmt. Der Rest des Programms wird dann für jede Permutation der Triade ABC genau einmal durchlaufen. Beim Permutationstyp 0 wird STØRE also nur einmal für die Triade ABC durchlaufen, bei 1 zweimal für ABC und BCA, und wenn schließlich die Relation "A" vom Permutationstyp 2 ist, wird STØRE dreimal durchlaufen für ABC, BCA und CAB. Durch Aufruf von ALØØK wird festgestellt, ob in F1 ein ID existiert, das zum ersten Element der abzuspeichernden Triaden-Permutation gehört, bei der Permutation ABC also das A-ID. Zur Vereinfachung der Beschreibung werden im Folgenden immer die Bezeichnungen A-ID, B-ID und C-ID verwendet; es wird also nur die Permutation ABC betrachtet. Es wird festgehalten, ob das A-ID gefunden wurde, und falls nicht, wird es durch Aufruf von AALLØC abgespeichert. Falls AALLØC meldet, daß kein Platz mehr auf dieser Triadenseite war, erfolgt Rücksprung von STØRE ins aufrufende Programm mit IRC=1. Durch Aufruf von IHASH wird die zum A-ID und B-ID gehörende Hash-Adresse bestimmt, und durch Aufruf von BRETRI wird festgestellt, ob das B-ID gefunden wurde. Falls ja, erfolgt die Abspeicherung von C im C-Modus, andernfalls von B und C im B-Modus.

Bei der Abspeicherung im C-Modus wird durch Aufruf von CRETRI festgestellt, ob das C-ID schon vorhanden ist; in diesem Falle ist die Triade ABC schon abgespeichert, und es erfolgt Rücksprung mit IRC=0 bzw. Abspeicherung der weiteren Permutationen. Als nächstes wird durch Aufruf von KEEPFR eine C-Zelle in F2 reserviert, die nachher das C-ID aufnehmen soll. Wenn KEEPFR meldet, daß keine C-Zelle mehr frei ist, erfolgt Rücksprung ins aufrufende

Programm mir IRC=1, nachdem gegebenenfalls ein erzeugtes A-ID wieder gelöscht wurde. Danach wird festgestellt, ob das gefundene B-ID ein Terminal war; in diesem Falle wird durch nochmaligen Aufruf von KEEPFR noch eine C-Zelle in F2 reserviert, in die die C-Information des Terminals übertragen wird. In das ehemalige Terminal wird dann stattdessen der Top-Zeiger des auf diese Weise neueröffneten C-Ringes eingetragen. Danach werden in jedem Falle die Zeiger des C-Ringes umgelenkt, um auch die neue C-Zelle in den Ring einzugliedern, und zwar als neue Bottom-Zelle. Dieser Vorgang wird im Folgenden als Update Old Bottom bezeichnet :



Dabei werden gleichzeitig die Ringpositions-Bits der einzelnen Zellen entsprechend abgeändert. Schließlich wird die neue Zelle als besetzte C-Zelle markiert und das C-ID eingetragen.

Bei der Abspeicherung von B und C im B-Modus wird zuerst festgestellt, ob die zum A-ID und zum B-ID gehörende Hash-Adresse eine leere Zelle bezeichnet; in diesem Falle können B und C sofort hier abgespeichert werden als Terminal. Da beim B-Modus das B-ID von BRETRI nicht gefunden worden war, handelt es sich in dem Falle, daß die Hash-Zelle nicht leer ist, um eine Conflict-Situation. Durch Aufruf von KEEPFR wird zunächst in F2 eine B-Zelle reserviert. Falls schon ein Conflict-Ring existiert, wird die neue Zelle durch

Update Old Bottom als neue Bottom-Zelle an diesen Ring angehängt, andernfalls wird sie zur Top- und Bottom-Zelle eines auf diese Weise eröffneten Conflict-Ringes gemacht. Danach wird in der A-Zelle in F1 nachgesehen, ob schon ein B-Ring existiert; falls nicht, wird die neue Zelle zur Top- und Bottom-Zelle eines so eröffneten B-Ringes gemacht. Andernfalls wird die Zelle durch Update Old Bottom zur neuen Bottom-Zelle des B-Ringes. In jedem Falle wird die Zelle schließlich als besetzte Terminal-Zelle markiert, und B-ID und C-ID werden eingetragen.

Wenn für jede geforderte Permutation die Triade abgespeichert wurde, ohne daß von AALLØC oder KEEPFR ein Reject produziert wurde, erfolgt Rücksprung ins aufrufende Programm mit IRC=0.

3.4.2.2 SUBROUTINE DELETE(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2,IRC) - Delete Triade

Länge : 58

wird aufgerufen von : DLRELE

Form des Aufrufs : CALL DELETE(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2,IRC)

Eingangsparameter : IDA1,IDA2 : ID der Relation "A"

IDB1,IDB2 : ID des ersten Entry "B"

IDC1,IDC2 : ID des zweiten Entry "C"

Ausgangsparameter : IRC : Reject-Code :

= 0 : alle Permutationen wurden
ordnungsgemäß gelöscht

= 1 : es wurde keine Triade gefunden

Transients : -

Externbezüge : EXDATA,ERRØR,ALØØK,IHASH,BRETRI,CRETRI

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : Wenn die Entity "A" keine Relation-Entity ist, erfolgt die Meldung
 ***** FALSCHE ENTITY-CLASS
 und Rücksprung ins aufrufende Programm mit
 IRC=1 .

Programmaufbau : Nachdem durch Aufruf von EXDATA sichergestellt ist, daß die Entity "A" eine Relation-Entity ist und ihr Permutationstyp bestimmt wurde, wird versucht, alle geforderten Permutationen der Triade ABC auf der aktuellen Seite des Triadenspeichers zu löschen. Falls dies gelang, erfolgt Rücksprung ins aufrufende Programm mit IRC=0, andernfalls mit IRC=1. Durch Aufruf von ALØØK wird festgestellt, ob das A-ID vorhanden ist; falls nicht, erfolgt Rücksprung. Durch Aufruf von IHASH wird die zum A-ID und zum B-ID gehörende Hash-Adresse bestimmt, und durch Aufruf von BRETRI wird festgestellt, ob das B-ID vorhanden ist; falls nicht, erfolgt Rücksprung. Schließlich wird durch Aufruf von CRETRI festgestellt, ob das C-ID vorhanden ist; falls nicht, erfolgt Rücksprung. Wenn die gefundene Zelle eine C-Zelle ist, ist sie ein Element eines C-Ringes, und die Triade wird durch Nullsetzen des C-ID als gelöscht markiert. Ist die gefundene Zelle eine B-Zelle, so ist sie ein Terminal, und die Triade wird durch Nullsetzen des B-ID und des C-ID als gelöscht markiert. Wenn jede geforderte Permutation der Triade auf diese Art gelöscht wurde, erfolgt Rücksprung ins aufrufende Programm mit IRC=0 .

3.4.2.3 SUBROUTINE RETRIV(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2,ICØD,IND) - Retrieve Triade

Länge : 48

wird aufgerufen von : FDRELE

Form des Aufrufs : CALL RETRIV(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2,
ICØD,IND)

Eingangsparameter : -

Ausgangsparameter : ICØD : Code, der angibt, ob eine Triade gefunden wurde :

- = 3 : es wurde ein C-Ring gefunden, dessen Elemente in eine Multi-Value-Entity abgespeichert wurden, deren ID als Antwort übergeben wird
- = 2 : es wurde ein C-Ring gefunden, der die gesuchte Triade ABC enthält, bzw. es wurde ein passender B-Ring mit Ringstart in F2 gefunden
- = 1 : es wurde ein passendes Terminal gefunden, bzw. es wurde ein passender B-Ring mit Ringstart in HAS gefunden
- = 0 : es wurde keine passende A-B-Assoziation gefunden
- = -1 : es wurde ein Terminal gefunden, das ABC nicht darstellt, bzw. B wurde nicht gefunden (ohne Conflict-Situation)
- = -2 : es wurde ein C-Ring gefunden, der keine passende Assoziation enthält, bzw. B wurde nicht gefunden (mit Conflict-Situation)

Die Werte von ICØD, die >0 sind, werden unverändert auch durch FDRELE ausgegeben.

IND : Index einer Zelle auf der aktuellen Triadenseite, und zwar bei :

ICØD=3 : Adresse des C-Ring-Bottom

ICØD=2 und ICØD=1 : Adresse des B-Ring-Top bzw. Adresse der gefundenen Zelle (C-Zelle bei ICØD=2 Terminal bei ICØD=1)

ICØD=0 : unbestimmt

ICØD=-1: Adresse des Terminals bzw.

Hash-Adresse von A und B, die von einem anderen als dem gewünschten B-ID besetzt ist

ICØD=-2: Adresse des C-Ring-Bottom bzw. Adresse des Bottom des Conflict Ringes, dessen Kopf die Hash-Adresse von A und B ist

Die Werte von IND werden unverändert auch durch FDRELE ausgegeben. Sie gelten für die Seite, die von FDRELE durch IPAGET angegeben wird.

Transients : IDA1,IDA2 : ID der Relation "A"

IDB1,IDB2 : ID des ersten Entry "B"

IDC1,IDC2 : ID des zweiten Entry "C"

IDs, nach denen gesucht werden soll, werden 0 gesetzt. Wurde nur ein ID nullgesetzt, so wird auf diesem ID, falls genau eine Triade gefunden wurde, das ID der gesuchten Entity zurückgegeben. Falls ein C-Ring gefunden wurde (ICØD=3), wird stattdessen das ID der durch PUTRIE erzeugten Multi-Value-Entity zurückgegeben, die die Elemente dieses Ringes enthält. Wurden zwei IDs nullgesetzt, so werden keine IDs (außer dem einen, das eingegeben wurde) zurückgegeben.

Externbezüge : PEØRDR,ALØØK,BRSTRT,REQCL1,PUTRIE

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Zuerst werden die eingegebenen IDs durch Aufruf von PEØRDR dem Fragetyp entsprechend richtig angeordnet, d.h. so, daß die Nullen hinten stehen. Dadurch werden die einzelnen Fragetypen folgendermaßen aufeinander abgebildet :

FO(ABC) → FO(ABC)

F1(ABO),F2(AOC),F3(OBC) → F1(ABO)

F4(AOO),F5(OOC),F6(OBO) → F4(AOO)

Somit werden nur noch drei Fragenklassen unterschieden : Klasse 0, bestehend aus der Frage FO nach der Existenz einer bestimmten Triade, Klasse 1, repräsentiert durch F1, der Frage, ob und in welchen Beziehungen zwei Entities zueinander stehen, und Klasse 2, repräsentiert durch F4, der Frage, ob es Entities gibt, die zu einer vorgegebenen Entity in einer Beziehung stehen, und wenn ja, in welcher. Durch Aufruf von ALØØK wird festgestellt, ob die Entity "A" existiert; falls nicht, erfolgt Rücksprung ins aufrufende Programm mit ICØD=0. Falls es sich um eine Frage der Klasse 2 handelt, wird dann durch Aufruf von BRSTRT das Top des B-Ringes als Index übergeben, und zwar mit ICØD=1, wenn es in HAS liegt, ICØD=2, wenn es in F2 liegt. Fragen der Klasse 1 werden dagegen durch Aufruf von REQCL1 beantwortet (siehe Abschnitt 3.5.2.3). Falls REQCL1 den Code ICØD=3 liefert, wird durch Aufruf von PUTRIE der durch die Adresse seines Top angegebene C-Ring in eine Multi-Value-Entity gepackt und deren ID bestimmt. Falls die Suche erfolgreich war, wird die bestimmte ID auf den Parameter-Positionen ausgegeben, die bei der Eingabe 0 waren.

3.4.2.4 SUBROUTINE GARCØL - Garbage Collection im Triadenspeicher

Länge : 273

wird aufgerufen von : SYSGCL

Form des Aufrufs : CALL GARCØL

Parameter : -

Externbezüge : GØBØTM,BKTIE,F2REL

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

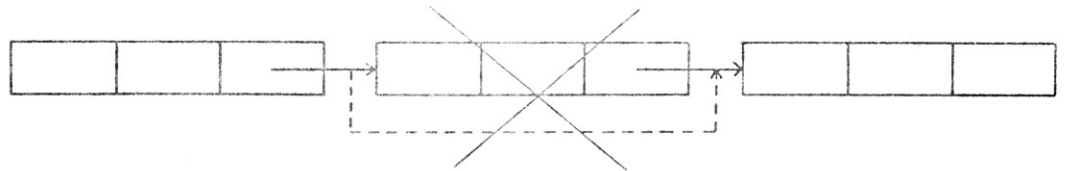
Programmaufbau : Die Garbage Collection, die den von gelöschten Triaden blockierten Speicherplatz wieder verfügbar macht, wird für die aktuelle Seite des Triadenspeichers ausgeführt. Sie läuft in 4 Schritten ab:

1. Die gelöschten C-Zellen in F2 werden freigegeben. Wird dabei ein ganz gelöschter C-Ring gefunden, dann wird die seinen Kopf bildende B-Zelle gelöscht.
2. Die gelöschten B-Zellen in F2, die ja sämtlich Elemente von Conflict-Ringen sind, werden freigegeben. Wird dabei ein ganz gelöschter B-Ring gefunden, dann wird die seinen Kopf bildende A-Zelle gelöscht.
3. Die gelöschten B-Zellen in HAS werden freigegeben. Wird dabei ein ganz gelöschter B-Ring gefunden, dann wird die seinen Kopf bildende A-Zelle gelöscht.
4. Die gelöschten A-Zellen in F1 werden freigegeben, d.h. aus dem occupied-Ring herausgenommen und in den vacant-Ring eingefügt.

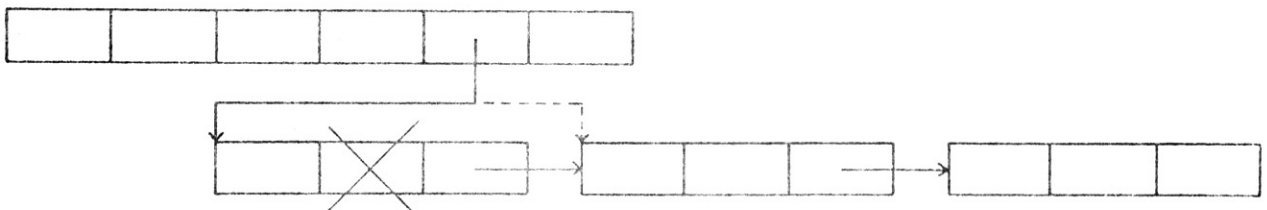
Diese 4 Schritte werden im Folgenden genauer beschrieben :

1. C-Collection in F2: Die Zellen von F2 werden in ihrer physikalischen Reihenfolge von vorn nach hinten abgearbeitet. Dabei werden leere, also im vacant-Ring liegende, Zellen ebenso

wie B-Zellen und besetzte, aber nicht gelöschte Zellen übersprungen. Die Behandlung erfolgt für Top-Zellen und Nicht-Top-Zellen getrennt und verschieden. Bei Nicht-Top-Zellen wird durch Aufruf von GØBØTM und BKTIE die im C-Ring links von der freizugebenden Zelle stehende Zelle bestimmt. Ist die gelöschte Zelle Bottom und die andere Top, so besteht der Ring nur aus diesen beiden Zellen. Sind beide Zellen gelöscht, so wird die B-Zelle, die Kopf dieses Ringes ist, gelöscht, und beide Zellen werden durch Aufruf von F2REL freigegeben. Ist die Top-Zelle des 2-Elemente-Ringes dagegen nicht gelöscht, so wird die B-Zelle, die Kopf des Ringes ist, zum Terminal gemacht, und die beiden C-Zellen werden durch Aufruf von F2REL freigegeben. Hat der Ring dagegen mehr als zwei Elemente, dann wird das sys-tie der links von der zu löschenden Zelle stehenden Zelle auf die rechts davon stehende Zelle gerichtet, und die Zelle wird durch Aufruf von F2REL freigegeben.



Bei Top-Zellen, die einem 2-Elemente-Ring angehören, wird wieder die angegebene Sonderbehandlung durchgeführt; bei allen anderen wird durch Aufruf von GØBØTM der Kopf des C-Ringes bestimmt, und dort wird als neuer Top-Zeiger der Zeiger auf den rechten Nachbarn der zu löschenden Zelle eingetragen, die dadurch zum neuen Ring-Top wird. Die gelöschte Zelle wird anschließend durch Aufruf von F2REL freigegeben.



2. B-Collection in F2 : Die Zellen von F2 werden wieder in ihrer physikalischen Reihenfolge von vorn nach hinten abgearbeitet. Dabei werden leere Zellen, C-Zellen und besetzte, aber nicht gelöschte Zellen übersprungen. Da alle B-Zellen in F2 zu Conflict-Ringen gehören, müssen für jede Zelle B-Ring und Conflict-Ring dieser Zelle geändert werden. Zuerst erfolgt das Update des Conflict-Ringes, wieder für Top-Zellen und Nicht-Top-Zellen getrennt. Bei Nicht-Top-Zellen wird durch Aufruf von GØBØTM und BKTIE die im Conflict-Ring links von der zu löschenden Zelle stehende Zelle bestimmt, und ihr con-tie wird auf die rechts davon stehende Zelle gerichtet. Bei Top-Zellen, die gleichzeitig Bottom-Zellen sind, besteht der ganze Conflict-Ring aus dieser einen Zelle; er kann somit durch Nullsetzen des con-tie der Kopf-Zelle ganz gelöscht werden. Bei allen anderen Top-Zellen wird durch Aufruf von GØBØTM der Kopf des Conflict-Ringes bestimmt, und dort wird als neuer Top-Zeiger der Zeiger auf den rechten Nachbarn der zu löschenden Zelle eingetragen.

Als nächster Schritt wird das Update des B-Ringes in genau der gleichen Weise durchgeführt, mit dem einzigen Unterschied, daß die Ring-Köpfe jetzt nicht B-Zellen in HAS, sondern A-Zellen in F1 sind. Schließlich wird die gelöschte Zelle durch Aufruf von F2REL freigegeben.

3. B-Collection in HAS : Die Zellen von HAS werden ebenfalls in ihrer physikalischen Reihenfolge von vorn nach hinten abgearbeitet. Dabei werden leere Zellen und besetzte, aber nicht gelöschte Zellen übersprungen. Nachdem das Update des B-Ringes in der be-

kannten Weise durchgeführt wurde, wird festgestellt, ob die Zelle Kopf eines Conflict-Ringes ist; falls nicht, wird sie durch einfaches Nullsetzen freigegeben. Wenn ein Conflict-Ring existiert, wird seine Top-Zelle in die zu löschende B-Zelle eingeschrieben, und das auf diese Zelle weisende sys-tie ihres B-Ringes wird umgelenkt. Wenn der Conflict-Ring nur aus dieser einen Zelle bestand, wird das con-tie des neuen Conflict-Ring-Kopfes gelöscht und der Conflict-Ring damit aufgelöst. Andernfalls wird die ehemalige zweite Zelle des Conflict-Ringes zur neuen Top-Zelle. Die auf diese Weise freigewordene Conflict-Zelle wird dann, da sie ja in F2 liegt, durch Aufruf von F2REL freigegeben.

4. A-Collection in F1 : Der occupied-Ring von F1 wird vom Top abwärts der Reihe nach abgearbeitet; dabei werden nicht gelöschte Zellen übersprungen. Der auf die freizugebende Zelle weisende Zeiger (der Top-Zeiger des Startregisters bei der Top-Zelle bzw. das sys-tie der linken Nachbarzelle bei allen anderen Zellen) wird auf die rechts von der gelöschten Zelle stehende Zelle gerichtet. Die gelöschte Zelle wird als leer deklariert, indem ihr erstes Wort nullgesetzt wird, und wird als neues Top vorne in den vacant-Ring eingefügt.

Diese Garbage-Collection-Routine kann zu jedem beliebigen Zeitpunkt, zu dem keine Routine aktiv ist, die den Triadenspeicher benötigt, aufgerufen werden. Es ist jedoch empfehlenswert, sie nur dann aufzurufen, wenn sich größere Mengen gelöschter Triaden angesammelt haben, da sich sonst der mit ihrem Aufruf verbundene Zeitaufwand (schätzungsweise 100 ms) nicht lohnt.

3.4.3 Gruppe S3 (Speicherinitialisierung)

3.4.3.1 SUBROUTINE NSINIT - Namenspeicher-Initialisierung

Länge : 13

wird aufgerufen von : SYSINT

Form des Aufrufs : CALL NSINIT

Parameter : -

Externbezüge : -

Common : /INAME/IN(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Die aktuelle Seite des Namenspeichers wird initialisiert, wobei der Systemparameter IN(2047), die Adresse der ersten Zelle der Conflict-Area dieser Seite, vor Aufruf der Routine schon im COMMON stehen muß. Dieser Wert wird dann auch als IN(2048), also als Adresse des Anfangs der vacant-Kette, übernommen. Nachdem die ganze Seite gelöscht wurde, wird die vacant-Kette in der Conflict-Area aufgebaut. Die Zellen werden in ihrer physikalischen Reihenfolge in der Kette hintereinander gehängt, wobei das erste Wort der ersten Zelle den Index IN(2047), das erste Wort der letzten Zelle den Index 2033 hat. Jede Zelle umfaßt 8 Worte, und jede Zelle der Conflict-Area hat in ihrem 8. Wort die Adresse des ersten Wortes der nächsten Zelle.

Anmerkung : IN(2047) muß der Bedingung

$$IN(2047) \equiv 1 \pmod{8}$$

genügen. Diese Bedingung wird nicht abgecheckt, da NSINIT im allgemeinen über SYSINT und nicht direkt vom Anwender aufgerufen wird.

3.4.3.2 SUBROUTINE F1INIT - Initialisierung des freien Speichers F1

Länge : 16

wird aufgerufen von : SYSINT

Form des Aufrufs: CALL F1INIT

Parameter : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Der freie Speicher F1 der aktuellen Seite des Triadenspeichers wird initialisiert, wobei der Systemparameter IP(2047), die Adresse des letzten Wortes von F1, vor Aufruf der Routine schon im COMMON stehen muß. Nachdem F1 gelöscht wurde, wird durch Setzen der Anfangsbedingung IP(1)=1 ein leerer occupied-Ring definiert. Danach werden die Zellen von F1 in ihrer physikalischen Reihenfolge als vacant-Ring organisiert, wobei das erste Wort der ersten Zelle den Index 3 hat, was durch Setzen des Startregisters IP(2)=3 angegeben wird, während das erste Wort der letzten Zelle, des Bottom, den Index IP(2047)-2 hat. Jede Zelle umfaßt 3 Worte und enthält in ihrem 3. Wort die Adresse des ersten Wortes der nächsten Zelle.

Anmerkung : IP(2047) muß der Bedingung

$$IP(2047) \equiv 2 \pmod{3}$$

genügen. Diese Bedingung wird nicht abgecheckt, da F1INIT im allgemeinen über SYSINT und nicht direkt durch den Anwender aufgerufen wird.

3.4.3.3 SUBROUTINE HSINIT - Initialisierung der Hash-Area HAS

Länge : 9

wird aufgerufen von : SYSINT

Form des Aufrufs : CALL HSINIT

Parameter : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Die Hash-Area HAS der aktuellen Seite des Triaden-
speichers wird initialisiert, wobei die System-
parameter IP(2047), die Adresse des letzten Wortes
von F1, und IP(2048), die Adresse des letzten
Wortes von HAS, vor Aufruf der Routine schon im
COMMON stehen müssen. Da HAS keine Ringstrukturen
enthält, genügt, wie bei der Hash-Area des
Namenspeichers, ein völliges Löschen zur Initiali-
sierung

Anmerkung : IP(2047) und IP(2048) müssen den Bedingungen
IP(2047)≡2mod3 und IP(2048)≡2mod3

genügen. Diese Bedingungen werden nicht abgecheckt ,
da HSINIT im allgemeinen über SYSINT und nicht
direkt durch den Anwender aufgerufen wird.

3.4.3.4 SUBROUTINE F2INIT - Initialisierung des freien Speichers F

Länge : 15

wird aufgerufen von : SYSINT

Form des Aufrufs : CALL F2INIT

Parameter : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Der freie Speicher F2 der aktuellen Seite des Triadenspeichers wird initialisiert, wobei der Systemparameter IP(2048), die Adresse des letzten Wortes von HAS, vor Aufruf der Routine schon in COMMON stehen muß. Nachdem F2 gelöscht wurde, wird in dem Wort mit dem Index IP(2048)+1 das Startregister des vacant-Ringes eingerichtet, der dann so aufgebaut wird, daß er die Zellen von F2 in ihrer physikalischen Reihenfolge enthält. Jede Zelle umfaßt 3 Worte und enthält in ihrem 3. Wort die Adresse des ersten Wortes der nächsten Zelle. Das erste Wort der ersten Zelle hat den Index IP(2048)+2, das erste Wort der letzten Zelle, des Bottom, hat den Index 2044.

Anmerkung : IP(2048) muß der Bedingung

$$IP(2048) \equiv 2 \pmod{3}$$

genügen. Diese Bedingung wird nicht abgecheckt, da F2INIT im allgemeinen über SYSINT und nicht direkt durch den Anwender aufgerufen wird.

3.4.3.5 SUBROUTINE DSINIT - Datenspeicher-Initialisierung

Länge : 19

wird aufgerufen von : SYSINT

Form des Aufrufs : CALL DSINIT

Parameter : -

Externbezüge : -

Common : /IENTI/IE(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Die aktuelle Seite des Datenspeichers wird initialisiert. Nachdem durch IE(258)=259 der Anfang des Datenbereichs festgelegt und dieser Bereich gelöscht wurde, wird die Ringstruktur am Anfang der Seite aufgebaut. Durch Einrichten des Startregisters IE(1)=1 wird ein leerer occupied-Ring definiert und die physikalisch erste Zelle dieses Teiles der Seite zum Top des vacant-Ringes erklärt. Nachdem dann der Ringbereich gelöscht wurde, werden seine Zellen in ihrer physikalischen Reihenfolge als vacant-Ring organisiert, wobei die erste Zelle aus den Worten mit den Indizes 2 und 3 und die letzte aus den Worten mit den Indizes 256 und 257 besteht. Jede Zelle umfaßt 2 Worte und enthält in ihrem 2. Wort die Nummer der nächsten Zelle. Dabei ist die Nummer einer Zelle gerade die Hälfte des Index ihres ersten Wortes, so daß diese Nummern, die ja der zweite Teil eines ID sind, jeweils nur 7 Bit haben.

3.4.4 Gruppe S4 (Ausgabe von Texten und Fragen)

3.4.4.1 SUBROUTINE ASK(I,N,ITEXT,M,IFELD) - Fragen an den Anwender

Länge : 27

wird aufgerufen von : CHNAME,DECSYN,DLSET,DLRELE,FDELEM,FDRELE,
RDNAME,STENT

Form des Aufrufs : INTEGER IFELD(M)
CALL ASK(I,N,NH...,M,IFELD)
Dabei ist NH... eine Hollerith-Konstant von
N Characters.

Eingangsparameter : N : Anzahl der in der ersten Zelle auszuge-
benden Zeichen (Characters).

ITEXT : INTEGER-Feld, auf dem eine Hollerith-
Konstante übergeben wird

M : Dimension des Feldes IFELD,
 = 0 : es wird keine zweite Zeile ausgegeben; IFELD wird nicht berücksichtigt. (Das Feld IFELD muß aber trotzdem als Parameter angegeben werden.)
 IFELD : Feld, dessen Inhalt in der zweiten Zeile in umcodierter Form ausgegeben wird

Ausgangsparameter : -

Transients : I : Eingang : = 0 : es wird eine Frage gestellt; die erhaltene Antwort wird durch den Ausgangswert von I mitgeteilt.
 ≠ 0 : es wird nur eine Meldung ausgegeben und keine Antwort erwartet.
 Ausgang : nur relevant, wenn der Eingangswert von I=0 war :
 = 1 : die beiden ersten Buchstaben der Eingaben waren JA oder ØK
 = 0 : sonst

Externbezüge : ERRØR,DØUTCD,DINCØD

Common : /BUFFER/IBUF(80),LBUF

Fehlerstops : Wenn der in einer der beiden Zeilen auszugebene Text größer als die Bufferlänge (siehe Abschnitt 3.2.1.1) ist, erfolgt Fehlerstop mit der Meldung
 ***** FRAGETEXT ZU LANG

Ausgegebene Texte : Es werden zwei Zeilen ausgegeben. In der ersten steht die auf der Parameter-Position von ITEXT angegebene Hollerith-Konstante, in der zweiten der als Zeichenkette (Character-String) interpretierte Inhalt des Feldes IFELD. Zum Beispiel bewirkt der Aufruf
 CALL ASK(1,11,11HNAME LAUTET,3,NAME),
 wenn NAME(1)=3595, NAME(2)=7691 und NAME(3)=7424 ist, was die Intern-Darstellung des Wortes DATAS ist, die Ausgabe der beiden folgenden Zeilen :
 ----- NAME LAUTET
 ----- DATAS

Programmaufbau : Nachdem die Länge der auszugebenden Zeilen überprüft wurde, wird die erste Zeile auf der logischen Gerätenummer 4 ausgegeben. Falls $M \neq 0$ ist, wird dann das Feld IFELD durch Aufruf von DØUTCD umcodiert und in den Buffer IBUF übertragen. Die Ausgabe erfolgt dann als zweite Zeile wieder auf der logischen Gerätenummer 4. Falls $I=0$ eingegeben wurde, wird eine Antwort auf der logischen Gerätenummer 8 eingelesen und durch Aufruf von DINCØD umcodiert. Wenn die beiden ersten Zeichen dieser Antwort JA oder ØK waren, wird $I=1$ gesetzt, andernfalls gleich 0. Wenn bei der Eingabe der Antwort ein Fehler gemacht wurde, wird die Frage von neuem gestellt.

3.4.4.2 SUBROUTINE ERRØR(N,ITEXT) - Fehlermeldungen und Fehlerstops

Länge : 12

wird aufgerufen von : CHNAME,DECSYN,STELEM,DLELEM,DLSET,STRELE,DLRELE,RDENT,RDSYN,FDELEM,LIELEM,FDRELE,FDRTØP,FDNEXT,SYSINT,STØRE,DELETE,RDNAME,PUTRIE,STENT,ALCNAM,DELENT,ASK,DSHIFT,DINCØD,DØUTCD,PAGE,DIALØG

Form des Aufrufs : CALL ERRØR(N,NH....)

Dabei ist NH.... eine Hollerith-Konstante von N Zeichen

Eingangsparameter : N : Anzahl der auszugebenden Zeichen und Steuer-Parameter (siehe unten)

ITEXT : INTEGER-Feld, auf dem eine Hollerith-Konstante übergeben wird.

Ausgangsparameter : -

Transients : -

Externbezüge : DUMPN,DUMP,DUMPE

Common : -

Fehlerstops : Wenn $N < 400$ ist, erfolgt nach Abarbeiten der Routine ein STØP. Diese Bedingung wird benutzt, um durch Aufruf von ERRØR von beliebigen Stelle im System aus einen STØP mit gewünschter Textausgabe und eventuell auch einem Dump erzeugen zu können.

Ausgegebene Texte : Es wird die auf der Parameter-Position von ITEXT angegebene Hollerith-Konstante ausgegeben. Zum Beispiel bewirkt der Aufruf

```
CALL ERRØR(6,6HFEHLER)
```

einen Fehlerstop mit der Ausgabe des Textes

```
***** FEHLER
```

Programmaufbau : Wenn N zwischen 100 und 199 angegeben wurde, wird durch Aufruf von DUMPN ein Dump der aktuellen Seite des Namenspeichers ausgegeben, bei N zwischen 200 und 299 durch Aufruf von DUMP ein Dump der aktuellen Seite des Triadenpeichers, und bei N zwischen 300 und 399 schließlich durch Aufruf von DUMPE ein Dump der aktuellen Seite des Entityspeichers. Nach Ausgabe der Hollerith-Konstanten auf der logischen Gerätenummer 4 erfolgt bei $N \geq 400$ Rücksprung ins aufrufende Programm, andernfalls STØP. Die Länge der Hollerith-Konstante wird in der Zehner- und Einer-Stelle von N angegeben :

```
CALL ERRØR(423,23HILLEGALER LØESCHVERSUCH)
```

3.5 Die Hilfsroutinen der Ebene 2

3.5.1 Gruppe H1 (Programme, die mit Namen- und Datenspeicher arbeiten)

3.5.1.1 SUBRØUTINE ALCNAM(NAME, IPAGE, IADR, ID1, ID2, IØC) -
Allocate Name

Länge : 29

wird aufgerufen von : STENT

Form des Aufrufs : INTEGER NAME(5), PAGTAB(109)

CALL ALCNAM(NAME, IPAGE, IADR, ID1, ID2, IØC)

Eingangsparameter : NAME : Feld, das den abzuspeichernden Namen linksbündig enthält

IPAGE : Nummer der Seite im Namenspeicher, auf der der Name abzuspeichern ist

IADR : Adresse der Zelle, in die die Information abzuspeichern ist, wenn die Zelle leer ist bzw. die Adresse der Endzelle einer Conflict-Kette, an die diejenige Zelle anzuhängen ist, in die die Information abzuspeichern ist

ID1, ID2 : zusammen mit dem Namen abzuspeicherndes ID

Ausgangsparameter : -

Transients : IØC : Ein- und Ausgang : = 0 : Zelle ist leer und kann zum Abspeichern benutzt werden
 = 1 : Zelle ist besetzt; die abzuspeichernde Information ist in eine andere Zelle abzuspeichern, die als Verlängerung der Conflict-Kette dieser Zelle anzuhängen ist.

Ausgang : = 2 : die vacant-Kette ist leer; auf dieser Seite können keine Conflict-Fälle mehr abgespeichert werden.

Externbezüge : ERRØR

Common : /DT/PAGTAB/INAME/IN(2048)

Fehlerstops : -

Ausgegebene Texte : Wenn ein Conflict-Fall vorliegt und die vacant-Kette leer ist, erfolgt die Meldung

***** NAMENSPEICHER VØLL

und Rücksprung ins aufrufende Programm mit IØC=2.

Programmaufbau : Wenn $I\emptyset C=0$ angegeben wurde, wird der Name NAME in die ersten 5 Worte der beim Index IADR beginnenden Zelle übertragen, und das ID wird in das 6. und 7. Wort der Zelle eingetragen. Bei $I\emptyset C=1$ wird abgefragt, ob $IN(2048)\neq 0$, die vacant-Kette also nicht leer ist. Bei nicht leerer vacant-Kette wird der in $IN(2048)$ stehende Index ihres Anfangs als Conflict-Information in die Zelle $IN(IADR+7)$ übertragen, wodurch die erste Zelle der vacant-Kette an die durch IADR adressierte Zelle und deren Conflict-Kette angehängt wird. Nachdem das Startregister auf die zweite Zelle der vacant-Kette gerichtet wurde und, falls es diese Zelle nicht mehr gibt, die Seite in der Seitentabelle als besetzt vermerkt wird, wird die neue Zelle durch Nullsetzen ihres 8. Wortes aus der vacant-Kette herausgelöst. Anschließend werden Name und ID in die ersten 7 Worte der Zelle eingetragen.

3.5.1.2 SUBROUTINE HASHNM(NAME,IPAGE,IADR) - Hash Name

Länge : 20

wird aufgerufen von : DELENT,RTRNID

Form des Aufrufs : INTEGER NAME(5),PAGTAB(109)
CALL HASHNM(NAME,IPAGE,IADR)

Eingangsparameter : NAME : Name, dessen Hash-Adresse zu bestimmen ist

Ausgangsparameter : IPAGE : Nummer der Seite, auf der der Name abzuspeichern ist

IADR : Hash-Index des Namens, Adresse des Anfangs der Conflict-Kette, an die der Name anzuhängen ist

Transients : -

Externbezüge : IE $\emptyset$ R,PAGE

Common : /DT/PAGTAB/INAME/IN(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Als Seitennummer IPAGE wird zunächst eine Zahl bestimmt, die als bitweises Exklusiv-Oder aus den beiden Zahlen

$$I1=CH2+32*CH1 \text{ und } I2=CH1+8*CH3+32*CH4$$

gebildet wird, wobei durch CHX das X-te Zeichen (Character) des Namens bezeichnet wird. Die so gebildete Zahl wird durch Restklassenbildung modulo der Anzahl der Seiten des Namenspeichers zur echten Seitennummer IPAGE. Durch Aufruf von PAGE wird diese Seite in den Kernspeicher geholt, und es wird festgestellt, wieviele Nicht-Conflict-Zellen auf dieser Seite vorhanden sind. Der Hash-Index IADR wird mit genau demselben Algorithmus, nur daß die Restklassenbildung jetzt modulo der Anzahl der Zellen geschieht, aus vier Pseudo-Zeichen berechnet, die aus allen Zeichen des Namens nach folgender Vorschrift berechnet werden :

$$J1 = \sum_{I=1}^5 \text{NAME}(I) + I1$$

$$J2 = I2 ; J2 = \text{NAME}(6-I) + J2 + \text{NAME}(I) \bmod 4 \text{ für } I=1, \dots, 5$$

$$CH1 = J1 \div 256, \quad CH2 = J1 \div CH1$$

$$CH3 = J2 \div 256, \quad CH4 = J2 \div CH3$$

Dabei bedeutet mod4 Restklassenbildung modulo 4 und $\div$ ganzzahlige Division

Anmerkung : Dieser Hash-Algorithmus wurde ohne theoretische Begründung experimentell aufgestellt und beibehalten, weil er völlig zufriedenstellend arbeitet (d.h. die Namen werden etwa gleichförmig verteilt, und es treten relativ wenig Conflict-Situationen auf). Die bisherige Erfahrung mit DATAS hat auch gezeigt, daß die Geschwindigkeit des Systems praktisch unabhängig vom

verwendeten Hash-Algorithmus ist, solange dieser einigermaßen vernünftig ist. Die Bearbeitungszeiten für die Berechnung von Hash-Funktionen und die Behandlung von Conflict-Situationen macht nur einen sehr geringen Bruchteil der für andere Operationen benötigten Zeit aus, so daß dort durch sinnvolle Programmierung erheblich mehr Zeit gespart werden kann als durch einen ausgeklügelten Hash-Algorithmus. Ferner ist das System so ausgelegt, daß ohne weitere Änderung jeder beliebige Hash-Algorithmus verwendet werden kann (siehe auch Abschnitt 3.5.3.2)

3.5.1.3 SUBROUTINE EXDATA(ID1,ID2,IDATA,ICLASS) - Extract Data

Länge : 14

wird aufgerufen von : STØRE,DELETE

Form des Aufrufs : CALL EXDATA(ID1,ID2,IDATA,ICLASS)

Eingangsparameter : ID1,ID2 : ID der Entity, deren Klasse und Permutationstyp gesucht werden

Ausgangsparameter : IDATA : bei einer Relation-Entity der Permutationscode (siehe Abschnitt 3.3.4.2)

ICLASS : Klasse der Entity :

=1 : Relation-Entity

=2 : Single-Value-Entity

=3 : Multi-Value-Entity

Transients : -

Externbezüge : PAGE

Common : /IENTI/IE(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Nachdem die richtige Seite des Datenspeichers durch Aufruf von PAGE in den Kernspeicher geholt wurde, wird die Klasse der Entity bestimmt. Bei Single-Value-Entities und Multi-Value-Entities erfolgt dann gleich Rücksprung ins aufrufende Programm, bei Relation-Entities erst, nachdem auch noch der Permutationstyp bestimmt wurde.

3.5.2 Gruppe H2 (Programme, die mit dem Triadenspeicher arbeiten)

3.5.2.1 SUBROUTINE KEEPFR(MØD,IND) - Keep Free

Länge : 43

wird aufgerufen von : STØRE

Form des Aufrufs : CALL KEEPFR(MØD,IND)

Eingangsparameter : MØD : Modus der zu reservierenden Zelle :
 =0 : B-Modus, Reservierung einer B-Zelle
 ≠0 : C-Modus, Reservierung einer C-Zelle

Ausgangsparameter : IND : Index des ersten Wortes der reservierten Zelle
 =0 : der vacant-Ring von F2 ist leer;
 auf dieser Seite können keine
 Conflict-B-Zellen und eventuell
 auch keine C-Zellen mehr abgespeichert
 werden

Transients : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Es wird eine Zelle im gewünschten Modus im freien Speicher F2 der aktuellen Seite des Triadenspeichers reserviert. Dazu wird zunächst durch Abfrage des Startregisters des vacant-Ringes von F2 festgestellt, ob noch 3-Wort-Zellen in F2 frei sind; falls nicht, erfolgt Rücksprung ins aufrufende Programm mit IND=0. Bei MØD≠0, also C-Modus, wird das Startregister auf das zweite Element des vacant-Ringes gerichtet, und der alte Wert des Startregisters wird als IND ausgegeben. Bei MØD=0, also B-Modus, wird ein Paar von Zellen aus dem vacant-Ring gesucht, die physikalisch direkt hintereinander liegen. Beide Zellen werden aus dem vacant-Ring herausgelöst und zu einer B-Zelle zusammengeschlossen, indem der Zeiger der ersten 3-Wort-Zelle auf die zweite gerichtet wird. Falls kein physikalisch benachbartes Paar von freien Zellen gefunden wurde, erfolgt wieder Rücksprung mit IND=0, andernfalls wird die Adresse des ersten Wortes der neuen 6-Wort-Zelle als IND ausgegeben.

3.5.2.2 SUBROUTINE AALLØC(IDA1,IDA2,IND) - A-Allocate

Länge : 16

wird aufgerufen von : STØRE

Form des Aufrufs : CALL AALLØC(IDA1,IDA2,IND)

Eingangsparameter : IDA1,IDA2 : in F1 abzuspeicherndes ID

Ausgangsparameter : IND : Adresse des ersten Wortes der A-Zelle,
in der das ID abgespeichert wurde,
=0 : der vacant-Ring von F1 ist leer;
auf dieser Seite können keine A-
Zellen mehr abgespeichert werden.

Transients : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Es wird eine A-Zelle im freien Speicher F1 der aktuellen Seite des Triadenspeichers reserviert, und in dieser wird das angegebene ID abgespeichert. Zunächst wird festgestellt, ob die vacant-Kette von F1 leer ist, d.h. ob ihr Startregister IP(2) auf sich selbst zeigt. In diesem Falle erfolgt ein Rücksprung ins aufrufende Programm mit IND=0. Andernfalls wird die Top-Zelle des vacant-Ringes aus diesem herausgelöst und mit dem abzuspeichernden ID geladen. Schließlich wird sie als neues Top in den occupied-Ring eingefügt, und die Adresse ihres ersten Wortes wird als IND ausgegeben.

3.5.2.3 SUBROUTINE REQCL1(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2,ICØD,IADR)
- Request Class 1

Länge : 15

wird aufgerufen von : RETRIV

Form des Aufrufs : CALL REQCL1(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2,
ICØD,IADR)

Eingangsparameter : IDA1,IDA2 : ID der A-Entity der gesuchten
oder zu überprüfenden Triade
IDB1,IDB2 : ID der B-Entity
IDC1,IDC2 : ≠0 : ID der C-Entity,
=0 : es ist eine Triade zu suchen,
die die Frage F1(ABO) beantwortet

Ausgangsparameter : ICØD : Code, der angibt, ob eine Triade gefunden wurde :
=3 : Fragestellung F1 : es wurde ein C-Ring gefunden

- =2 : Fragestellung FO(ABC): es wurde ein C-Ring gefunden
- =1 : Fragestellung F1 und FO : es wurde ein Terminal gefunden
- =0 : es wurde keine passende A-B-Assoziation gefunden
- =-1; Fragestellung FO : es wurde ein Terminal gefunden, das C nicht enthält;
bzw. Fragestellung F1 und FO :
B wurde nicht gefunden (ohne Conflict)
- =-2: Fragestellung FO : es wurde ein C-Ring gefunden, der C nicht enthält; bzw. Fragestellung F1 und FO :
B wurde nicht gefunden (mit Conflict)

Bei der Fragestellung FO bzw. einer Frage der Klasse 1(F1,F2,F3, siehe Abschnitt 3.4.2.3) werden die Werte von ICØD unverändert auch durch RETRIV ausgegeben.

- IND : Index einer Zelle auf der aktuellen Triadenseite, und zwar bei
- ICØD=3: Adresse des C-Ring-Top
 - ICØD=2: Adresse der gefundenen C-Zelle
 - ICØD=1: Adresse des gefundenen Terminals
 - ICØD=0: unbestimmt
 - ICØD=-1 : Adresse des Terminals bzw.
Hash-Adresse von A und B, die von einem anderen als dem gewünschten B-ID besetzt ist.
 - ICØD=-2 : Adresse des C-Ring-Bottom
bzw. Adresse des Bottom des Conflict-Ringes, dessen Kopf die Hash-Adresse von A und B ist.

Die Werte von IND werden bei der Fragestellung FO bzw. einer Frage der Klasse 1 unverändert auch durch RETRIV ausgegeben.

Transients : -

Externbezüge : IHASH,ALØØK,BRETRI,CRETRI

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Durch Aufruf von IHASH,ALØØK und BRETRI wird festgestellt, ob eine passende A-B-Assoziation vorhanden ist. Falls nicht, erfolgt Rücksprung ins aufrufende Programm; andernfalls wird durch Aufruf von CRETRI eine passende B-C-Assoziation gesucht und der Index einer gefundenen C-Zelle bzw. eines gefundenen C-Ring-Top bei der Fragestellung F1 bestimmt, wobei der eventuell von CRETRI gelieferte Code-3 durch +3 ersetzt wird.

3.5.2.4 SUBROUTINE PUTRIE(ID1,ID2,IND) - Put Ring Into Entity

Länge : 85

wird aufgerufen von : RETRIV

Form des Aufrufs : CALL PUTRIE(ID1,ID2,IND)

Eingangsparameter : -

Ausgangsparameter : ID1,ID2 : ID der erzeugten Multi-Value-Entity:
=0, wenn keine erzeugt wurde

Transients : IND : Eingang : Top des C-Ringes, der in eine
Multi-Value-Entity zu füllen ist
Ausgang : Index des letzten Elements, das in
die Multi-Value-Entity gefüllt wurde,
d.h. Index des C-Ring-Bottom bei
höchstens 100 nicht gelöschten Ele-
menten im Ring, bzw. Index des 100.
nicht gelöschten Elementes des Ringes
sonst

Externbezüge : PAGE,ERRØR,RTRNID,STENT

Common : //IP(2048)/IENTI/IE(2048)/ISTACK(201)

Fehlerstops : -

Ausgegebene Texte : Wenn der abzuspeichernde C-Ring mehr als 100 Elemente hat, erfolgt die Meldung
 ***** UEBER 100 IDS. NUR 100 UEBERTRAGEN
 Wenn nach dem unten beschriebenen Verfahren kein Name mehr erzeugt werden kann, erfolgt die Meldung
 ***** KEIN NAME FREI
 und Rücksprung ins aufrufende Programm. Dabei wird als IND der Index der ersten nicht gelöschten Zelle des C-Ringes und als ID null ausgegeben.

Programmaufbau : Zunächst wird das ID des C-Ring-Top bestimmt und festgestellt, ob diese Zelle gelöscht ist; falls ja, wird das ID der nächsten Zelle auf dem Ring stattdessen bestimmt. Wenn auf diese Art festgestellt wurde, daß der ganze C-Ring gelöscht ist, erfolgt ohne weitere Meldung Rücksprung ins aufrufende Programm und als ID wird 0 übergeben. Andernfalls wird das ID der ersten nicht gelöschten Zelle auf dem C-Ring festgehalten und nach einem Aufruf von PAGE der Name der zugehörigen Entity bestimmt. Daraus wird der Name der zu erzeugenden Multi-Value-Entity in folgender Weise hergestellt : Die ersten acht Characters des gefundenen Namens werden zu den letzten acht Characters des Namens der Multi-Value-Entity, wobei ihre Reihenfolge beibehalten wird. Davor werden dann die beiden Zeichen *O gesetzt. Später wird durch Aufruf von RTRNID festgestellt, ob dieser Name schon existiert; falls ja, werden die beiden ersten Zeichen durch *1 ersetzt, und so weiter, über *9, *A, *B bis *Z. Sind alle diese Namen schon vorhanden, so erfolgt Fehlermeldung und Rücksprung ins aufrufende Programm, wobei 0 als ID zurückgegeben wird. Wurde zum Beispiel der

Name LØKØMOTIVE gefunden und wurde festgestellt, daß der Name *OLØKØMØTI schon existiert, so wird der Name *1LØKØMØTI erzeugt. Nach dem Herstellen des Namens werden maximal die ersten 100 nicht gelöschten IDs in das Feld ISTACK übertragen, wobei der Index IND mitläuft. Dann wird durch Aufruf von RTRNID festgestellt, ob der hergestellte Name schon existiert; falls ja, wird durch Aufruf von PAGE die zugehörige Seite des Datenspeichers in den Kernspeicher geholt. Es wird festgestellt, ob die gefundene Entity eine Multi-Value-Entity ist, ob sie genau so viele Elemente hat wie die neu zu erzeugende, und ob das dieselben Elemente in derselben Reihenfolge sind. Werden alle diese Fragen bejaht, so erfolgt Rücksprung ins aufrufende Programm, ohne daß eine neue Entity erzeugt wurde, und als ID wird das ID der gefundenen Entity übergeben. Wird eine der Fragen mit nein beantwortet, so wird der nächste Name erzeugt und festgestellt, ob eine Entity dieses neuen Namens existiert. Diese wird dann wieder mit der zu erzeugenden Entity verglichen, und so weiter, bis entweder eine Entity gefunden ist, die genau gleich der zu erzeugenden Entity ist, oder bis keine Namen mehr erzeugt werden können oder bis ein Name erzeugt wurde, der noch nicht vorhanden war. In diesem letzteren Fall wird die erzeugte Entity durch Aufruf von STENT abgespeichert und ihr ID durch Aufruf von RTRNID bestimmt. Durch diesen etwas umfangreichen Algorithmus wird erreicht, daß einerseits verschiedene Mengen erzeugt werden können, auch wenn das Startelement dasselbe ist, was zum Beispiel bei Änderung der Triadenstruktur wichtig sein kann, und daß andererseits nie eine schon vorhandene Menge noch einmal erzeugt wird. Die so erzeugte Multi-Value-Entity enthält als Elemente alle nicht gelöschten Elemente des C-Ringes bis zu einer maximalen Anzahl von 100 in

derselben Reihenfolge, in der sie im Ring stehen.

3.5.2.5 SUBROUTINE BRSTRT(INDA,ICØD,ISTART)- B-Ring-Start

Länge : 10

wird aufgerufen von : RETRIV

Form des Aufrufs : CALL BRSTRT(INDA,ICØD,ISTART)

Eingangsparameter : INDA : Index der A-Zelle, deren zugehöriges
B-Ring-Top bestimmt werden soll

Ausgangsparameter : ICØD : Code, der angibt, in welchem Speicher-
bereich der Ring-Start liegt :
=1: die Top-Zelle liegt in HAS
=2: die Top-Zelle liegt in F2
ISTART : Index des B-Ring-Tops

Transients : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : In der durch INDA angegebenen A-Zelle wird
nachgesehen, welchen Index das zugehörige
B-Ring-Top hat, und je nach dessen Wert wird
ICØD festgesetzt.

3.5.2.6 SUBROUTINE BKTIE(IHEAD,IND,LEFT,IPØS) - Back Tie

Länge : 28

wird aufgerufen von : GARCØL

Form des Aufrufs : CALL BKTIE(IHEAD,IND,LEFT,IPØS)

Eingangsparameter : IHEAD : Adresse des Ring-Kopfes
 IND : Adresse der Zelle, deren Vorgänger
 im Ring bestimmt werden soll
 IPØS : Position des sys-tie; Code, der an-
 gibt, um welche Art von Ring es sich
 handelt :
 =0: B-Ring
 =1: C-Ring
 =2: Conflict-Ring

Ausgangsparameter : LEFT : Adresse des linken Ringnachbarn der
 Zelle mit dem Index IND

Transients : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Nachdem durch IPØS der Typ des Ringes bestimmt
 wurde, wird aus IHEAD die Adresse des sys-tie
 bzw. con-tie des Ring-Top bestimmt. Danach wird
 der Ring nach dem Vorgänger der durch IND adres-
 sierten Zelle abgesucht und deren Index bestimmt.
 Ein Abchecken, ob IND die Top-Zelle des Ringes
 ist, erfolgt nicht, da die Routine nur von
 GARCØL und nicht vom Anwender direkt aufgerufen
 wird.

3.5.2.7 SUBROUTINE F2REL(MØD,IND) - F2-Release

Länge : 23

wird aufgerufen von : GARCØL

Form des Aufrufs : CALL F2REL(MOD,IND)

Eingangsparameter : MØD : Modus der freizugebenden Zelle :
 =0 : B-Modus, Freigabe einer B-Zelle
 ≠0 : C-Modus, Freigabe einer C-Zelle

IND : Adresse des ersten Wortes der freizugebenden Zelle

Ausgangsparameter : -

Transients : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Die durch IND adressierte Zelle wird gelöscht und vorn in den vacant-Ring von F2 eingefügt. Dazu werden das erste und zweite und bei B-Zellen auch das vierte und fünfte Wort der Zelle gelöscht. Das Startregister des vacant-Ringes wird auf das erste Wort gerichtet, und bei B-Zellen wird das dritte Wort gleich dem Index des vierten gesetzt, wodurch die B-Zelle in zwei 3-Wort-Zellen aufgelöst ist. Schließlich wird in das letzte Wort der Zelle der Index des ehemaligen Top des vacant-Ringes eingetragen.

3.5.2.8 SUBROUTINE BRETRI(IADR,IDB1,IDB2,INDA,ICØD) - B-Retrieval

Länge : 44

wird aufgerufen von : STØRE,DELETE,REQCL1

Form des Aufrufs : CALL BRETRI(IADR,IDB1,IDB2,INDA,ICØD)

Eingangsparameter : IDB1,IDB2 : ID der zu suchenden B-Entity
INDA : Index der A-Entity, deren Assoziation mit der B-Entity festgestellt werden soll.

Ausgangsparameter : ICØD : Code, der angibt, ob eine A-B-Assoziation gefunden wurde :
=2 : B wurde auf einem Conflict-Ring (also in F2) gefunden

- =1 : B wurde in der Zelle in HAS gefunden,
die durch den Eingangswert von IADR
adressiert ist; keine Information über
die Existenz eines Conflict-Ringes
- =0 : die durch den Eingangswert von IADR
adressierte Zelle ist leer
- =-1: B steht nicht in der durch den Eingangs-
wert von IADR adressierten Zelle in
HAS, und zu dieser Zelle existiert auch
kein Conflict-Ring
- =-2: B steht nicht in der durch den Eingangs-
wert von IADR adressierten Zelle in HAS;
diese Zelle ist Kopf eines Conflict-
Ringes, der B nicht enthält

Unter "B" ist hier immer ein B-ID zu verstehen,
das mit dem in der durch INDA adressierten Zelle
stehenden A-ID assoziiert ist.

Transients : IADR : Eingang : aus dem A-ID und dem B-ID gebil-
dete Hash-Adresse

Ausgang : ICØD=2 : Adresse des B-ID; IADR
wurde geändert

ICØD=1 : Adresse des B-ID; IADR
unverändert

ICØD=0 : IADR unverändert

ICØD=-1: IADR unverändert

ICØD=-2: Adresse des Bottom des
Conflict-Ringes, dessen
Kopfzelle durch den Ein-
gangswert von IADR adressiert
wurde; IADR wurde geändert

Externbezüge : GØBØTM

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Wenn die durch IADR adressierte Zelle leer ist, erfolgt Rücksprung ins aufrufende Programm mit $IC\emptyset D=0$. Wenn in der durch IADR adressierten Zelle das angegebene B-ID steht, wird durch Aufruf von $G\emptyset B\emptyset TM$ und anschließenden Vergleich der Adresse des B-Ring-Kopfes mit INDA festgestellt, ob Assoziation mit A vorliegt; falls ja, erfolgt Rücksprung mit $IC\emptyset D=1$. Andernfalls wird, genau wie in dem Fall, daß ein anderes als das angegebene B-ID in der durch IADR adressierten Zelle steht, festgestellt, ob die Zelle Kopf eines Conflict-Ringes ist; falls nein, erfolgt Rücksprung mit $IC\emptyset D=-1$. Andernfalls wird der Conflict-Ring nach dem angegebenen B-ID abgesucht, und bei jedem gefundenen passenden B-ID erfolgt der oben beschriebene Hash-Check. Wurde der ganze Ring ohne Erfolg abgesucht, so erfolgt Rücksprung mit $IC\emptyset D=-2$; andernfalls wird die Suche abgebrochen, sobald sie Erfolg hatte, und es erfolgt Rücksprung mit $IC\emptyset D=2$.

3.5.2.9 SUBROUTINE CRETRI(IADR, IDC1, IDC2, ICØD) - C-Retrieval

Länge : 33

wird aufgerufen von : STØRE,DELETE,REQCL1

Form des Aufrufs : CALL CRETRI(IADR, IDC1, IDC2, ICØD)

Eingangsparameter : IDC1, IDC2 : ID der zu suchenden C-Entity
=0 : es ist nur der Index des
C-Ring-Top zu bestimmen

Ausgangsparameter : -

Transients : IADR : Eingang : Index der B-Zelle, zu der eine C-Assoziation zu bestimmen ist

Ausgang : ICØD=2 : Adresse des C-ID,IADR
wurde geändert

ICØD=1,0,-1 : IADR unverändert

ICØD=-2 : Adresse des Bottom des C-Ringes, dessen Kopfzelle durch den Eingangswert von IADR adressiert wurde; IADR wurde geändert

ICØD=-3 : Adresse des Top des C-Ringes dessen Kopfzelle durch den Eingangswert von IADR adressiert wurde; IADR wurde geändert.

ICØD : Eingang : >0: B-Retrieval hatte Erfolg; C-Retrieval ist durchzuführen

≤0: B-Retrieval hatte keinen Erfolg, C-Retrieval ist nicht durchzuführen

Ausgang : =2 : B ist Kopf eines C-Ringes, der C enthält

=1 : B ist Terminal und enthält C

=0 : Suche wurde nicht durchgeführt, weil Eingangswert von ICØD=0

=-1: B ist Terminal und enthält C nicht

=-2: B ist Kopf eines C-Ringes, der C nicht enthält

=-3: Sonderfall für REQCL1, wenn als ID null angegeben wurde: B ist Kopf eines C-Ringes

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Wenn ICØD≤0 angegeben wurde, erfolgt Rücksprung ins aufrufende Programm. Dann wird abgefragt, ob die durch den Eingangswert von IADR adressierte B-Zelle Terminal ist; falls ja, wird das in ihr enthaltene C-ID mit dem angegebenen ID verglichen, und je nach dem Ergebnis dieses Vergleichs er-

folgt Rücksprung ins aufrufende Programm mit ICØD=1 oder ICØD=-1. Andernfalls wird die Adresse des C-Ring-Top bestimmt, und falls als ID null angegeben wurde, erfolgt Rücksprung mit ICØD=-3. Wenn das angegebene ID von 0 verschieden war, wird der C-Ring nach einer Zelle abgesucht, die dieses ID enthält, und je nach dem Ergebnis dieser Suche erfolgt Rücksprung mit ICØD=2 oder ICØD=-2.

3.5.2.10 SUBROUTINE ALØØK(IDA1,IDA2,IND) - A-Look

Länge : 16

wird aufgerufen von : STØRE,DELETE,RETRIV,REQCL1

Form des Aufrufs : CALL ALØØK(IDA1,IDA2,IND)

Eingangsparameter : IDA1,IDA2 : ID der zu suchenden A-Zelle

Ausgangsparameter : IND : Index der gefundenen A-Zelle,
 =0 : die Zelle wurde auf dieser Seite
 des Triadenspeichers nicht gefunden

Transients : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Die in den Zellen des occupied-Ringes der aktuellen Seite des Triadenspeichers enthaltenen IDs werden der Reihe nach mit dem angegebenen ID verglichen. Bei Übereinstimmung wird die Suche abgebrochen, und der Index der gefundenen Zelle wird ausgegeben. Wird dagegen die Bottom-Zelle des occupied-Ringes erreicht, ohne daß das angegebene ID gefunden wurde, so wird als Index 0 ausgegeben.

3.5.2.11 SUBROUTINE GØBØTM(IND,IPØS) - Go to Bottom

Länge : 10

wird aufgerufen von : STØRE,DELETE,GARCØL,BRETRI

Form des Aufrufs : CALL GØBØTM(IND,IPØS)

Eingangsparameter : IPØS : Position des Ring-tie in den Zellen
des Ringes

= 0 : sys-tie im 2. Wort der Zelle;
B- oder C-Ring

= 1 : con-tie im 5. Wort der Zelle;
Conflict-Ring

Ausgangsparameter : -

Transients : IND : Eingang : Index einer B- oder C-Zelle
Ausgang : Index der zugehörigen Bottom-Zelle

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Nachdem aus IPØS die Position des Ring-tie innerhalb der Zelle bestimmt wurde, also insbesondere bei B-Zellen entschieden wurde, ob die Bottom-Zelle ihres B-Ringes oder ihres Conflict-Ringes gesucht werden soll, wird abgefragt, ob die Zelle schon Bottom ist. In diesem Falle wird ihr Index ausgegeben, andernfalls wird sie durch die nächste Zelle auf dem Ring ersetzt, und das Verfahren wird wieder angewendet.

3.5.3 Gruppe H3 (Programme ohne Speicherbezug)

3.5.3.1 SUBROUTINE PEØRDR(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2) -
Permutation Order of Request

Länge : 26

wird aufgerufen von : RETRIV

Form des Aufrufs : CALL PEØRDR(IDA1,IDA2,IDB1,IDB2,IDC1,IDC2)

Eingangsparameter : -

Ausgangsparameter : -

Transients : Eingang : IDA1,IDA2 : ID der Relation
 IDB1,IDB2 : ID des ersten Entry
 IDC1,IDC2 : ID des zweiten Entry

Ausgang : IDA1,IDA2 : ID der A-Entity
 IDB1,IDB2 : ID der B-Entity
 IDC1,IDC2 : ID der C-Entity

IDs, nach denen durch RETRIV gesucht werden soll,
 werden 0 gesetzt. Diese IDs werden durch PEØRDR
 durch zyklische Permutation der IDs nach hinten
 sortiert.

Externbezüge : -

Common : -

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Die eingegebenen IDs werden zyklisch so per-
 mutiert, daß alle nullgesetzten IDs nachher
 am Schluß der Parameterliste stehen. Dadurch
 wird die in Abschnitt 3.4.2.3 angegebene Ein-
 teilung der möglichen Fragen in drei Frage-
 klassen erzielt :

Klasse 0 : FO(ABC) → FO(ABC)

Klasse 1 : F1(ABO), F2(AOC), F3(OBC) → F1(ABO)

Klasse 2 : F4(AOO), F5(OOC), F6(OBO) → F4(AOO)

3.5.3.2 FUNCTION IHASH(IDA1,IDA2,IDB1,IDB2) - Hash Triaden

Länge : 13

wird aufgerufen von : STØRE,DELETE,REQCL1

Form des Aufrufs : INTEGER PAGTAB(109)

IADR=IHASH(IDA1,IDA2,IDB1,IDB2)

Eingangsparameter : IDA1,IDA2 : ID der A-Entity
 IDB1,IDB2 : ID der B-Entity

Ausgangsparameter : Funktionswert IHASH : aus den beiden IDs
 gebildete Hash-Adresse für
 die aktuelle Seite des
 Triadenspeichers

Transients : -

Externbezüge : IEØR

Common : //IP(2048)/DT/PAGTAB

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Aus den IDs werden zunächst zwei Zahlen IDA und
 IDB berechnet, die die Information aller vier
 Parameter enthalten :

$ID = ID2 + 128 * \text{Nummer der Datenspeicher-Seite}$
 Dabei werden die Seiten des Datenspeichers von 0
 an durchnummeriert. Nachdem aus IP(2047) und
 IP(2048) die Anzahl der Zellen in HAS bestimmt
 wurde (siehe Abschnitt 2.2.2.2), wird durch bit-
 weise Exklusiv-Oder-Verknüpfung von IDA und IDB
 eine Hash-Zahl berechnet, die durch Restklassen-
 bildung modulo der Anzahl der Zellen in HAS auf
 den Adressenraum von HAS abgebildet wird.

Anmerkung : Dieser Hash-Algorithmus wurde ohne theoretische Be-
 gründung experimentell aufgestellt und beibehalten,
 weil er bis jetzt zufriedenstellend zu arbeiten scheint;
 für eine genaue Beurteilung seiner Güte wurden bis
 jetzt noch zuwenig Erfahrungen mit DATAS gesammelt.
 Die genaue Form des Hash-Algorithmus ist für das
 System auch völlig unwichtig, da er jederzeit nach
 Bedarf durch einen anderen Algorithmus ersetzt werden
 kann, ohne daß sonstige Änderungen am System nötig

wären. Die bisherigen Erfahrungen zeigen allerdings, daß die für die Berechnung der Hash-Funktion und für die Bearbeitung von Conflict-Situationen benötigte Zeit aufgrund der Auslegung der Triadenprogramme für hohe Geschwindigkeit so gering ist, daß sich eine Optimierung der Hash-Funktion wahrscheinlich nicht lohnt, da zu erwarten ist, daß sie kaum noch Ersparnisse an Speicherplatz oder Rechenzeit bringt. Aus diesen Gründen und in Anbetracht der Untersuchung von Lum, Yuen und Dodd / 5 /, die für viele Anwendungen eine leichte Überlegenheit des primitiven Divisionsverfahren über andere kompliziertere Hash-Algorithmen ergab, wurde für den oben beschriebenen Algorithmus entschieden.

3.5.3.3 SUBROUTINE DINCØD(IFELD,ICØMT) - Data in Coding

Länge : 52

wird aufgerufen von : ASK,ALPHAD,DIALØG

Form des Aufrufs : INTEGER IFELD(40),ICØMT(25)
CALL DINCØD(IFELD,ICØMT)

Eingangsparameter : -

Ausgangsparameter : IFELD : Feld, auf das die umcodierten Daten zu schreiben sind

ICØMT : Feld, das die Struktur der Eingabezeile enthält :

ICØMT(I), $1 \leq I \leq 23$: Stellung des I-ten Kommas in der Zeile

ICØMT(24) : Anzahl der Kommas in der Zeile

ICØMT(25) : Anzahl der Worte, die die Zeile belegt,

>0 : die Zeile enthielt kein Apostroph

<0 : die Zeile enthielt ein Apostroph, bei dem die Umcodierung abgebrochen wurde

=0 : es trat ein Fehler auf

Transients : -

Externbezüge : ERRØR

Common : /BUFFER/IBUF(80),LBUF

Fehlerstops : -

Ausgegebene Texte : Wenn die Eingabezeile mehr als 23 Kommata enthält, erfolgt die Meldung
 ***** KOMMAFEHLER
 und Rücksprung ins aufrufende Programm mit ICØMT(25)=0. Wenn eines der Zeichen der Eingabezeile nicht zu dem vorgesehenen Zeichenvorrat
 _ (blank) 0...9 A...Z +-*/*.(),='
 gehört, erfolgt die Meldung
 ***** ZEICHEN NICHT IDENTIFIZIERBAR
 und Rücksprung mit ICØMT(25)=0

Programmaufbau : Nachdem die Felder IFELD und ICØMT gelöscht wurden, erfolgt die Umcodierung und Strukturuntersuchung der ersten LBUF Zeichen der Eingabezeile, die im Feld IBUF im COMMON stehen muß. Die Elemente von IBUF werden der Reihe nach mit den Elementen eines durch DATA mit Hollerithkonstanten vorbesetzten Feldes ICONT verglichen. Immer wenn eine Übereinstimmung festgestellt wurde, erhält das Element von IBUF die zugehörige Code-Zahl zugewiesen, die gleich der FORTRAN-Interndarstellung des Zeichens in der Varian V73 ist. Dazu werden die Zeichen in der oben angegebenen Reihenfolge von 0 bis 45 durchnumeriert, das hier als Steuerzeichen verwendete Apostroph wird dagegen auf die Code-Zahl 0, also Blank, abgebildet. Wurde an einer Stelle der Eingabezeile ein Komma gefunden, so wird seine Spaltennummer in ein Element des Feldes ICOMT übertragen und der Kommazähler ICOMT(24) erhöht. Wurde ein Apostroph gefunden, so wird die Umcodierung der Zeile an dieser Stelle abgebrochen, und der Wortzähler ICOMT(25)

wird mit einem negativen Vorzeichen versehen. Wurden mehr als 23 Kommata in der Eingabezeile gefunden oder wurde auf einer Position der Eingabezeile ein Zeichen gefunden, das mit keinem der in ICØMT abgespeicherten Zeichen übereinstimmt, so wird an dieser Stelle die Umcodierung abgebrochen, und es erfolgt Rücksprung ins aufrufende Programm mit ICØMT(25)=0. Schließlich werden die solchermaßen umcodierten Zeichen zu je zweien nach der Formel

$$\text{IFELD}(I)=256*\text{IBUF}(2*I-1)+\text{IBUF}(2*I)$$

in das Feld IFELD übertragen.

Anmerkung : Diese ziemlich umständliche Eingaberoutine kann bei der Varian V73 bis auf den Aufbau der Komma-Tabelle ICØMT entfallen, da schon beim normalen Einlesen der hier mühsam aufgebaute Code erzeugt wird. Bei den meisten Rechnern, speziell bei allen, die Character-Verarbeitung vorsehen, kann die Umcodierung erheblich einfacher und schneller erfolgen. Die gesamte Routine ist lediglich bei Rechnern mit einer Struktur wie der der Telefunken TR440 nötig, da hier durch die Verwendung einer Typenkennung und eines FØRTRAN-Compilers, der keine einfache Textverarbeitung erlaubt, ein umständliches Vergleichsverfahren zur Umcodierung leider nicht zu umgehen ist.

3.5.3.4 SUBRØUTINE DØUTCD(N,IFELD) - Data Out Coding

Länge : 20

wird aufgerufen von : ASK,DIALØG

Form des Aufrufs : INTEGER IFELD(N)
CALL DØUTCD(N,IFELD)

Eingangsparameter : N : Anzahl der umzucodierenden Worte
IFELD : Feld, das die auszugebende Zeile enthält

Ausgangsparameter : -

Transients : -

Externbezüge : ERRØR

Common : /BUFFER/IBUF(80),LBUF

Fehlerstops : Wenn $N > (LBUF+1)/2$ ist, erfolgt Fehlerstop mit der
Meldung

***** DATENSATZ ZU LANG

Ausgegebene Texte : -

Programmaufbau : Nachdem die Länge des Datensatzes überprüft wird, wird das Feld IBUF, das nachher die Ausgabezeile enthalten soll, mit Blanks vorbesetzt. Dann werden die einzelnen Worte des Feldes IFELD umcodiert und die dabei jeweils erhaltenen zwei Zeichen in zwei Worte des Feldes IBUF übertragen. Die auszugebenden Zeichen sind dabei, wie auch die von DINCØD erkennbaren Zeichen (siehe Abschnitt 3.5.3.3) in einem durch DATA vorbesetzten Feld ICØNT als Hollerithkonstanten enthalten. Zahlen der Interndarstellung, die in ICØNT kein Äquivalent haben, werden auf das Apostroph abgebildet.

Anmerkung : Diese Routine kann bei der Varian V73 entfallen, da die hier verwendete Zeichendarstellung genau dem FØRTRAN-Interncode der V73 entspricht. Bei Maschinen, die Characterverarbeitung zulassen, wie zum Beispiel bei der Control Data CD3300, kann die Routine durch eine erheblich einfachere und schnellere ersetzt werden. In der hier angegebenen Form ist sie nur bei Rechnern notwendig, die durch die Verwendung einer Typenkennung und verschiedener Abspeicherverfahren für INTEGER-Größen und Hollerithkonstanten eine schnelle Ein- und Ausgabe intern zu verarbeitender Zeichenketten nicht zulassen. Die Routine DØUTCD wird insbesondere also bei Einsatz auf der Telefunken TR440 benötigt.

3.5.4 Gruppe H4 (Dump-Programme)

3.5.4.1 SUBROUTINE DUMPN - Dump des Namenspeichers

Länge : 17

wird aufgerufen von : ERROR

Form des Aufrufs : CALL DUMPN

Parameter : -

Externbezüge : -

Common : /INAME/IN(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Der Inhalt der aktuellen Seite des Namenspeichers wird in Form von zwei Tabellen mit je 64 Zeilen und 16 Spalten im INTEGER-Format auf der logischen Gerätenummer 6 ausgegeben.

Anmerkung : Bei Maschinen, die eine oktale Zahlenausgabe erlauben, wie zum Beispiel bei der Control Data CD 3300 oder der digital pdp 11, kann ohne sonstige Änderungen das INTEGER-Format durch das OCTAL-Format ersetzt werden, wodurch der Dump erheblich übersichtlicher und leichter interpretierbar wird. Es ist vorgesehen, eventuell bei der Telefunken TR 440, die keine Ausgabe von Oktal-Zahlen vorsieht, ein Assemblerprogramm einzusetzen, das die Umcodierung in den BCD-Code unterdrückt und damit eine Ausgabe von Oktal-Zahlen doch ermöglicht.

3.5.4.2 SUBROUTINE DUMPE - Dump des Entityspeichers

Länge : 17

wird aufgerufen von : ERROR

Form des Aufrufs : CALL DUMPE

Parameter : -

Externbezüge : -

Common : /IENTI/IE(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Der Inhalt der aktuellen Seite des Datenspeichers wird in Form von zwei Tabellen mit je 64 Zeilen und 16 Spalten im INTEGER-Format auf der logischen Gerätenummer 6 ausgegeben.

Anmerkung : Hier gilt ebenfalls das in der Anmerkung von Abschnitt 3.5.4.1 Gesagte.

3.5.4.3 SUBROUTINE DUMP - Dump des Triadenspeichers

Länge : 17

wird aufgerufen von : ERRØR

Form des Aufrufs : CALL DUMP

Parameter : -

Externbezüge : -

Common : //IP(2048)

Fehlerstops : -

Ausgegebene Texte : -

Programmaufbau : Der Inhalt der aktuellen Seite des Triadenspeichers wird in Form von zwei Tabellen mit je 64 Zeilen und 16 Spalten im INTEGER-Format auf der logischen Gerätenummer 6 ausgegeben.

Anmerkung : Hier gilt wieder das in der Anmerkung von Abschnitt 3.5.4.1 Gesagte.

3.6 Voraussetzungen über den Inhalt der FØRTRAN-Bibliothek

Gruppe FB (Funktionen in der FØRTRAN-Bibliothek)

FUNCTION IEØR(I,J) - Exclusive Or

wird aufgerufen von : IHASH,HASHNM

Form des Aufrufs : K=IEØR(I,J)

Eingangsparameter : I,J : zwei INTEGER-Zahlen, als Bit-Ketten
interpretiert

Ausgangsparameter : Funktionswert IEØR : bitweise gebildetes
Exklusiv-Oder von I und J

Transients : -

erlaubte Externbezüge : -

erlaubter Common : -

Fehlerstops : -

ausgegebene Texte : -

Anmerkung : Das Vorhandensein einer INTEGER-Funktion, die zwei
INTEGER-Zahlen bitweise durch Exklusiv-Oder verknüpft,
in der FØRTRAN-Bibliothek des Rechners wird voraus-
gesetzt. Diese Funktion steht bei der Varian V73 nach
ISA-Norm unter dem Namen IEØR in der Support Library
des Betriebssystems VORTEX zur Verfügung, und für die
Control Data CD3300 wurde sie in Form eines CØMPASS-
Unterprogramms ebenfalls zur Verfügung gestellt. In
der FØRTRAN-Bibliothek der Telefunken TR440 ist die
Funktion unter dem Namen LØGAUT vorhanden, so daß
für die DATAS-Version dieses Rechners die entsprechen-
den Funktionsaufrufe umgeändert wurden.

4. Hinweise für den Gebrauch des Systems

4.1 Ausgegebene Meldungen

In diesem Abschnitt wird eine Liste der verschiedenen vom System ausgegebenen Meldungen gegeben, die für jede Meldung den Namen des Programms angibt, das diese Meldungen ausgibt, sowie die Nummer des Abschnittes, in dem dieses Programm beschrieben wird. Anhand des Kapitels 3 lassen sich daher leicht die Bedingungen für das Auftreten der einzelnen Meldungen finden.

4.1.1 Fehlerstops :

Diese Meldungen werden durch die Routine ERRØR auf der logischen Gerätenummer 4 ausgegeben. Sie sind durch die Zeichenkette ***** vor der Textzeile gekennzeichnet.

1. BUFFERLAENGE NICHT AUSREICHEND - SYSINT(3.3.9.1)
2. BUFFERLAENGE UEBERSCHRITTEN - SYSINT(3.3.9.1)
3. DATENSATZ ZU LANG - DØUTCD(3.5.3.4)
4. FALSCHER BUFFERLAENGE - DIALØG(3.2.1.1)
5. FALSCHER SEITENTYP - PAGE(3.3.9.2)
6. FALSCHER SEITENNUMMER - PAGE(3.3.9.2)
7. FEHLER BEI NAME-CØNFLICT - DELENT(3.4.1.2)
8. FEHLER BEI SYNØNYM-LOESCHUNG - DELENT(3.4.1.2)
9. FEHLER IM ALGØRITHMUS - DSHIFT(3.4.1.3)
10. FRAGETEXT ZU LANG - ASK(3.4.4.1)
11. SEITENLAENGE UEBERSCHRITTEN - SYSINT(3.3.9.1)
12. SYSTEMKAPAZITÄT UEBERSCHRITTEN - SYSINT(3.3.9.1)

4.1.2 Systemmeldungen

Diese Meldungen werden, abgesehen von den Meldungen des Programms DIALØG, die direkt auf der logischen Gerätenummer 9 ausgegeben werden, durch Aufruf der Routine ERRØR auf der logischen Gerätenummer 4 ausgegeben. Alle Systemmeldungen sind durch die Zeichenkette ***** vor der Textzeile gekennzeichnet.

1. BØTTØM SCHON ERREICHT - FDNEXT(3.3.8.3)
2. DATENBEREICH-UEBERSCHREITUNG - RDENT(3.3.5)
3. DATENBLØCK ZU LANG FUER FREIEN SPEICHER - STENT(3.4.1.1)
4. DATENSPEICHER VØLL - STENT(3.4.1.1)
5. ELEMENT EXISTIERT NICHT - FDELEM(3.3.7.1)
6. ENTITY EXISTIERT NICHT - RDENT(3.3.5),RDSYN(3.3.6.2)
7. FALSCHER ENTITY-CLASS - STØRE(3.4.2.1),DELETE(3.4.2.2)
8. FALSCHER PARAMETERANZAHL - DIALØG(3.2.1.1)
9. FALSCHER INDEX - FDRTØP(3.3.8.2),FDNEXT(3.3.8.3)
10. FALSCHER SEITENNUMMER - FDRTØP(3.3.8.2),FDNEXT(3.3.8.3)
11. FALSCHER ZAHLENEINGABE - DIALØG(3.2.1.1)
12. ID UNGUELTIG - RDNAME(3.3.6.1)
13. ILLEGALER LØESCHVERSUCH - DELENT(3.4.1.2)
14. ILLEGALER NAME - STENT(3.4.1.1)
15. ILLEGALER RELATIØNSNAME - STRELE(3.3.4.2)
16. KEINE ELEMENTBEZIEHUNG - DLELEM(3.3.3.3)
17. KEINE ENTITY DIESES NAMENS DA - CHNAME(3.3.2.1),DECSYN(3.3.2.2)
18. KEINE ID-INFØRMATION - RDNAME(3.3.6.1)
19. KEINE MENGE - FDELEM(3.3.7.1),LIELEM(3.3.7.2)
20. KEINE MULTI-VALUE-ENTITY - DLELEM(3.3.3.3),DLSET(3.3.3.4)
21. KEINE RELATIØN - DLRELE(3.3.4.3),FDRELE(3.3.8.1)
22. KEIN NAME FREI - PUTRIE(3.5.2.4)
23. KEIN SYNØNYM MIT DIESER NUMMER VØRHANDEN - RDNAME(3.3.6.1)
24. KØMMAFEHLER - DINCØD(3.5.3.3)
25. LEERE MENGE - DLELEM(3.3.3.3)
26. MENGE EXISTIERT NICHT - FDELEM(3.3.7.1),LIELEM(3.3.7.2)
27. MENGE KANN NICHT RELATIØN SEIN - STRELE(3.3.4.2)
28. MENGE NICHT VORHANDEN - DLELEM(3.3.3.3)
29. NAMENSPEICHER VØLL - ALCNAM(3.5.1.1)
30. NUMMER NICHT IM ELEMENTBEREICH - LIELEM(3.3.7.2)
31. NUR 100 ELEMENTE GELØESCHT - DLSET(3.3.3.4)
32. RELATIØN KANN NICHT MENGE SEIN - STELEM(3.3.3.2)
33. TRIADE GEFUNDEN - FDRELE(3.3.8.1)
34. TRIADE NICHT VORHANDEN - FDRELE(3.3.8.1)
35. TRIADENSPEICHER VØLL - STRELE(3.3.4.2)
36. TRIADE UNGUELTIG - FDRELE(3.3.8.1)
37. UEBER 100 IDS. NUR 100 UEBERTRAGEN - PUTRIE(3.5.2.4)

- 38. UEBERLAUF BEI ELEMENT-SPEICHERUNG - STELEM(3.3.3.2)
- 39. UEBERLAUF BEI RELATIØN-SPEICHERUNG - STRELE(3.3.4.2)
- 40. UEBERLAUF BEI SYNØNYM-SPEICHERUNG - STENT(3.4.1.1)
- 41. UNBEKANNTE ANWEISUNG - DIALØG(3.2.1.1)
- 42. UNGUELTIGES ID - DLSET(3.3.3.4)
- 43. ZEICHEN NICHT IDENTIFIZIERBAR - DINCØD(3.5.3.3)

4.1.3 Aufforderungen an den Dialog-Partner

Diese Meldungen werden vom Programm DIALØG(3.2.1.1) auf der logischen Gerätenummer 9 ausgegeben. Sie sind durch die Zeichenkette ===== vor der Textzeile gekennzeichnet.

- 1. BITTE ANWEISUNG EINGEBEN
- 2. BITTE BUFFERLAENGE EINGEBEN

Zwei weitere Aufforderungen werden vom Programm ALPHAD(3.2.2.2) auf der logischen Gerätenummer 9 ausgegeben. Sie sind ebenfalls durch die Zeichenkette ===== vor der (ersten) Textzeile gekennzeichnet:

- 3. BITTE NAECHSTEN DATENBLØCK EINGEBEN
 - 4. BITTE LETZTEN DATENBLOCK EINGEBEN
- MAX. LAENGE =

Durch Aufruf von ASK wird von den Routinen CHNAME(3.3.2.1), DECSYN(3.3.2.2) und STENT(3.4.1.1) aus der Anwender gefragt, ob eine Entity, deren Name angegeben wird, überspeichert werden darf. Es wird eine Antwort des Anwenders auf der logischen Gerätenummer 8 erwartet. Sind die beiden ersten Zeichen der Antwort JA oder ØK, so wird die Antwort als "Ja" gewertet, andernfalls als "Nein". Die Frage umfaßt zwei Textzeilen, die durch die Zeichenkette ----- vor dem Text gekennzeichnet sind, und auf der logischen Gerätenummer 4 ausgegeben werden. Die zweite Zeile enthält den Namen der Entity, die erste den folgenden Text:

- 5. UEBERSPEICHERN VØN

4.1.4 Meldungen über Entities

Diese Meldungen werden durch die Routine ASK auf der logischen Gerätenummer 4 ausgegeben. Sie umfassen jeweils zwei Textzeilen, die durch die Zeichenkette ----- vor dem Text gekennzeichnet sind. Eine Antwort wird nicht erwartet. Die zweite Zeile enthält den Namen der Entity, oder der Entities, die erste einen der folgenden Texte :

1. ELEMENTBEZIEHUNG ZWISCHEN - FDELEM(3.3.7.1)
2. KEINE ELEMENTBEZIEHUNG - FDELEM(3.3.7.1)
3. KEIN LØESCHEN VØN - DLSET(3.3.3.4)
4. NAME LAUTET - RDNAME(3.3.6.1)
5. NICHT VØRHANDEN- - DLRELE(3.3.4.3), FDRELE(3.3.8.1)

4.1.5 Sonstige Meldungen

Das Programm DIALØG(3.2.1.1) gibt bei einigen speziellen Befehlen noch Zusatzinformationen auf der logischen Gerätenummer 4 aus. Diese Meldungen sind nicht eigens gekennzeichnet. Im folgenden werden jeweils die Befehle, die die Ausgabe der Meldungen veranlassen, mit angegeben :

1. ANZAHL DER ELEMENTE =.... - LIE(3.2.1.3.8)
2. DATENLAENGE =..... KLASSE =..... - RDE(3.2.1.3.6)
3. SYN.ZAHL =..... SYN.NR.=..... - RDS(3.2.1.3.6)
4. TRIADENSEITE =,..... INDEX=..... - FDR(3.2.1.3.7), FDN(3.2.1.3.7)

4.2 Technische Voraussetzungen für den Einsatz von DATAS

4.2.1 Speicherplatz

Hier kann kein genereller Wert angegeben werden, da je nach der verwendeten Maschine und je nach der von der Maschine zur Verfügung gestellten Software (wie zum Beispiel FØRTRAN-Compiler, Relocatable Loader, Library Genator) erhebliche Unterschied im Umfang des erzeugten Objektprogrammes bestehen. So verlangt das System auf der Control Data CD 3300 bei 24 Bit Wortlänge 21k Kernspeicher, auf der Telefunken TR 440 dagegen bei 48 Bit Wortlänge

29k. Diese Zahlen können bei Verwendung von Overlays, die für den Einsatz auf der Varian V73 geplant sind, erheblich reduziert werden. Die Aufteilung in die einzelnen Overlays sollte sich dabei an der im Abschnitt 2.3.2 beschriebenen Verweisstruktur des Systems orientieren.

4.2.2 Wortlänge

Das System setzt eine minimale Wortlänge von 16 Bit voraus, da alle Daten in Form von positiven INTEGER-Zahlen von maximal 15 Bit Länge abgespeichert werden. Auf eine Verwendung des 16-ten Bits, des Vorzeichens-Bits also, wurde verzichtet, um das System nicht auf Maschinen einer bestimmten Wortlänge und einer bestimmten Darstellung negativer Zahlen festzulegen. Das System arbeitet in Bezug auf den Speicherplatz somit am rationellsten bei 16-Bit-Maschinen wie der V73. Durch das Setzen der Single Option beim Aufruf des FØRTRAN-Compilers kann bei den meisten Maschinen mit kurzer Wortlänge (z.B. 24 Bit bei der CD 3300) die Länge von INTEGER-Zahlen auf ein Wort beschränkt werden, bei Maschinen großer Wortlänge (z.B. IBM) auf ein Halbwort, so daß der verschenkte Speicherplatz relativ klein gehalten werden kann. Die TR 440 sieht leider eine solche generelle Beschränkung auf Halbworte nicht vor. Um die gerade bei dieser Maschine beträchtliche Speicherplatz-Verschwendung (nur 15 von 48 Bit werden gebraucht!) in einigermaßen erträglichen Grenzen zu halten, müßten alle Variablen einzeln als INTEGER*2, d.h. als Halbworte vereinbart werden. Da die Maschine aber keine Halbwort-Konstanten kennt, müßten bei jedem Aufruf von Unterprogrammen erhebliche Änderungen im Quellprogramm vorgenommen werden, die diese Aufrufe ziemlich umständlich und damit zeitraubend machen würden. Dadurch dürfte die Arbeitsgeschwindigkeit des Systems, die aufgrund eines ungünstigen Objektcodes bei der TR 440 nur um etwa den Faktor 1.8 höher liegt als auf der acht Jahre älteren CD 3300, noch erheblich absinken. Da dies nicht tragbar erscheint und da durch die Einführung von INTEGER*2-Größen das auf der TR 440 ohnehin schon sehr umfangreiche Objektprogramm noch weiter anwachsen dürfte, wurde auf die Einführung solcher Größen bisher verzichtet, zuungunsten der Größe von Namen-, Daten- und Triadenspeicher.

4.2.3 Externe Geräte und Dateien (Files), Ein- und Ausgabe

Das System kennt folgende logische Geräteummern :

- 4 : Parallele Ausgabe von Meldungen und Daten auf Terminal und Schnelldrucker
- 6 : Ausgabe von Dumps auf dem Schnelldrucker
- 8 : Eingabe von Befehlen und Daten vom Terminal aus
- 9 : Ausgabe von Aufforderungen an den Anwender auf das Terminal allein
- 10 : Datei auf einem externen Speichermedium, die die Seitentabelle sowie die Seiten von Namen-, Triaden- und Datenspeicher enthält

Diese Geräteummern entsprechen - mit Ausnahme der 10 - den an der Telefunken TR 440 üblichen. (Bei den Maintenance-Versionen 13 und 14 des Betriebssystems BS3, die bis jetzt in Saarbrücken gelaufen sind, werden allerdings die Nummern 4 und 9 aufgrund eines Fehlers in der FØRTRAN-Ein-/Ausgabe gleich behandelt: Bei ausgeschaltetem Druckerprotokoll bei Dialogbetrieb erscheinen keinerlei Ausgaben über diese Nummern auf dem Ausdruck. Ein Unschädlichmachen dieses Software-Fehlers war bis jetzt leider noch unmöglich.) Bei anderen Maschinen müssen gegebenenfalls diese Geräteummern umdefiniert werden, sofern nicht vorgezogen wird, die Nummern in den - relativ wenigen - READ- und WRITE-Anweisungen abzuändern.

Die in den Programmen DIALØG(3.2.1.1), ALPHAD(3.2.2.2) und ASK(3.4.4.1) verwendeten READ-Anweisungen enthalten einen ERR-Parameter, der bei Maschinen wie der Telefunken TR 440 oder der digital pdp11 verwendet wird, um Eingaben im falschen Format abzufangen. Bei Maschinen, deren FØRTRAN-Compiler keinen ERR-Parameter vorsieht, wie zum Beispiel bei der Control Data CD3300 oder der Varian V73, ist dieser Parameter aus den READ-Anweisungen zu entfernen.

Die unter der logischen Geräteummer 10 anzusprechende Datei ist vor dem Start des DATAS-Systems als Permanent-File bzw. für Testzwecke möglicherweise auch als Scratch-File mit belie-

bigem Namen und beliebigem Passwortschutz mit der folgenden Struktur anzulegen :

Typ : RAM : Random-Datei mit Zugriff über Satzmarke

Zugriffskoordinierung : P : Privatdatei, oder : G : Gemeinschaftsdatei

Satzzahl : Mn oder Un : maximal oder ungefähr n Sätze;
n ≥ Gesamtzahl der Seiten + 2

Satzbau : M2048W : maximal 2048 Worte

Der erste Satz der Datei enthält die Page-Tabelle, die folgenden die einzelnen Seiten von Namen-, Triaden- und Datenspeicher.

Die Sätze der Datei werden durch READ- und WRITE-Anweisungen mit direktem (Random-)Zugriff gelesen und beschrieben; bei Maschinen, deren FØRTRAN-Compiler keinen Direct-Access-Parameter in I/Ø-Anweisungen vorsieht, wie bei der Varian V73 oder der Control Data CD3300, müssen Positionierung und Ein- und Ausgabe-Vorgang getrennt vorgenommen werden.

5. Vergleich mit dem ursprünglichen Konzept

5.1 Definition des ursprünglichen Konzepts von DATAS

Die Entwicklung eines implementierungsfähigen Konzepts der assoziativen Datenstruktur DATAS / 11 / wurde als Diplomarbeit für Herrn Volker Glatzer am Lehrstuhl und Institut für Informationsverarbeitung der Technischen Universität Berlin von Herrn Prof.Dr.W. Giloi am 12. März 1971 vergeben und von Herrn Dr.-Ing. J. Encarnação betreut. Die in dem vorliegenden Bericht beschriebene DATAS-Implementierung stützt sich weitgehend auf diese Diplomarbeit / 6 /, aus der sie entwickelt wurde. Die genannte Diplomarbeit enthält ein Systemkonzept zur programmierungstechnischen Implementierung einer Assoziativspeicherung, die bei der graphischen Informationsverarbeitung (Computer Graphics) und im Rahmen des Information Retrieval einsetzbar ist. In der Arbeit werden die damals als Schnittstelle Benutzer-DATAS geplanten Programme

```

RETURN_ID (entspricht RTRNID)
DECLARE_CLASS (ohne Entsprechung)
STØRE_SINGLE (≐STSNGL)
CHANGE_SINGLE (≐CHSNGL)
STØRE_MULTI (≐STELEM mit Einschränkungen)
STØRE_TRIADE (≐STØRE)
DELETE_TRIADE (≐DELETE)
PUT_RING_INTØ_ENTITY (≐PUTRIE)
PERM_ØRDER_ØF_REQUEST (≐PEØRDR)
REQUEST_CLASS_I (≐REQCL1)

```

kurz beschrieben. Anschließend folgt eine Beschreibung des geplanten Triadenspeichers. Den Hauptteil der Arbeit bilden genaue Beschreibungen mit Flußdiagrammen der geplanten Programme

```

F1_INITIAL (≐F1INIT)
A_LØØKUP (≐ALØØK)
A_ALLØCATE (≐AALLØC)
F2_INITIAL (≐F2INIT)
KEEP_FREE (≐KEEPFR)
GØ_TØ_BØTTØM (≐GØBØTM)
B_RETRIEVAL (≐BRETRI)

```

```

C_RETRIEVAL(≐CRETRI)
STØRE_TRIADE(≐STØRE)
DELETE(≐DELETE)
PUT_RING_INTØ_ENTITY(≐PUTRIE)
PERM_ØRDER_ØF_REQUEST(≐PEØRDR)
REQUEST_CLASS_I(≐REQCL1)
B_RING_START(≐BRSTRT)
RETRIEVAL_PROGRAM(≐RETRIV)
BACK_TIE_ØN_CØNFLICT(≐BKTIE mit Einschränkungen)
F2_RELEASE(≐F2REL)
F1_GARBAGE_CØLLECTION }
F2_GARBAGE_CØLLECTION } (≐GARCØL)
HAS_GARBAGE_CØLLECTION }

```

Den Rest der Arbeit bildet eine Beschreibung des geplanten Datenspeichers und der darin abgespeicherten Entities, sowie des geplanten Paging-Systems und der zur Handhabung dieser Größen nötigen Programme :

```

RETURN_ID(≐RTRNID)
DECLARE_CLASS(ohne Entsprechung)
STØRE_SINGLE(≐STSNGL)
CHANGE_SINGLE(≐CHSNGL)
STØRE_MULTI(≐STELM mit Einschränkungen)

```

Die vorliegende DATAS-Implementierung wurde aus diesem Konzept entwickelt; die Änderungen gegenüber dem Konzept, die sich im Laufe der Implementierung zwangsläufig ergaben oder als zweckmäßig erscheinen und deshalb eingebaut wurden, werden in den folgenden Abschnitten diskutiert.

5.2 Entities

5.2.1 Wortlänge und IDs

Das ursprüngliche Konzept war für einen Einsatz auf einer Rechenanlage des Typs SDS 930 (CAE 90-40) mit einer Wortlänge von 24 Bit geplant. Daher konnten IDs von 19 Bit Wortlänge für die Adressierung von je max. 2^{10} Entities auf max. 2^9 Seiten vorgesehen werden, und die restlichen Bit konnten für Erweiterungen des Systems vorbehalten bleiben. Die hier beschriebene Implementierung

wurde jedoch speziell im Hinblick auf einen Einsatz des DATAS-Systems auf einer Varian V73 mit 16 Bit Wortlänge geplant. Da 16 Bit für einen universellen Einsatz des Systems als zu wenig erschienen, wurden zwei Worte für die Abspeicherung der IDs vorgesehen, so daß zur Zeit 18 Bit zur Verfügung stehen, um je max. 2^7 Entities auf max. 2^{11} Seiten zu adressieren; eine Erweiterung auf 22 Bit ist bei geringfügiger Änderung der Struktur der A-Zellen in F1 möglich. Die in der vorliegenden DATAS-Implementierung benutzte Wortlänge von 15 Bit macht daher erhebliche Änderungen in der Struktur der Zellen des Triaden-speichers notwendig. Diese Änderungen werden im Abschnitt 5.3 beschrieben.

5.2.2 Entity-Klassen

Im ursprünglichen Konzept war ebenfalls eine Einteilung der Entities in die drei Klassen der Relation-Entities, der Single-Value-Entities und der Multi-Value-Entities vorgesehen. Das genaue Format der einzelnen Entities sollte jedoch durch die Definition von Unterklassen formatgleicher Entities durch die Routine DECLARE_CLASS bestimmt werden. Die Definition einer Entity im ursprünglichen Konzept von DATAS setzte somit die vorherige Definition einer zugehörigen Klassen-Entity voraus, die eine Menge formatgleicher Entities repräsentierte, so daß jede Entity als Element einer Menge aufgefaßt wurde. Da für den Entity-Speicher jedoch eine formatfreie Abspeicherung der Entities vorgesehen wurde und da Mengen ebenso gut durch Multi-Value-Entities repräsentiert werden können, konnte auf die Einführung solcher Klassen-Entities und auf die Funktion DECLARE_CLASS verzichtet werden. Diese Änderung wurde hauptsächlich zur Bequemlichkeit des Anwenders eingeführt, ebenso wie die automatische Umdefinition der Entity-Klasse durch STRELE und STELEM.

5.2.3 Adressierung der Entities

Zur Bequemlichkeit des Anwenders wurde das Hauptgewicht der Adressierung von Entities von den IDs auf die Namen verlegt, wobei die im ursprünglichen Konzept noch vorgesehenen Schlüssel als Zwischengrößen zwischen Namen und ID entfielen. Eine wahlweise Adressierung über Namen und/oder IDs wurde durch

Vorsehen der Routinen RTRNID und RDNAME, die ein beliebiges Wechseln zwischen beiden Adressierungsweisen ermöglichen, dem Anwender freigestellt. Als zusätzliche Erweiterung wurde die Möglichkeit von Synonymen vorgesehen.

5.3 Triaden

5.3.1 Einteilung der Seiten des Triadenspeichers

Die ursprünglich vorgesehene Einteilung der Seiten des Triadenspeichers in die Bereiche F1,HAS und F2 wurde beibehalten, die Adressen der einzelnen Worte wurden jedoch nicht mehr auf die einzelnen Bereiche, sondern auf die ganze Seite bezogen, was speziell eine Unterscheidung zwischen HAS und F2 durch das ursprünglich vorgesehene Area-Bit überflüssig machte. Außerdem konnten dadurch alle Abfragen, auf welchen Bereich sich ein bestimmter Index bezieht, weggelassen werden, so daß die meisten Triaden-Programme erheblich gekürzt und damit auch schneller gemacht werden konnten.

5.3.2 Struktur der A-, B- und C-Zellen

Durch die vorgesehene kürzere Wortlänge war eine Erhöhung der Anzahl der Worte, die die einzelnen Zellentypen belegen, von 2 auf 3 bzw. von 4 auf 6 notwendig geworden. Dabei wurde bei B- und C-Zellen lediglich das für die Abspeicherung des bzw. der IDs vorgesehene Wort durch (je) zwei Worte ersetzt. Dabei wurde (aus ästhetischen Gründen) das Modus-Bit vom ersten in das zweite Wort der Zelle verlagert. Weiterhin entfiel das Area-Bit, das durch die abgeänderte Adressierungsweise überflüssig geworden war. Da sich bei A-Zellen sys-tie und B-pointer nicht in einem Wort unterbringen ließen, eine Aufteilung auf zwei Worte aber wegen der Länge des ID vier Worte für jede A-Zelle erfordert hätte, wurde die in Abschnitt 2.2.2.2 beschriebene, etwas komplizierte Aufteilung der A-Zellen gewählt, wobei wieder das Area-Bit entfallen konnte.

5.3.3 Bit-Manipulationen

Im ursprünglichen Konzept wurden für die Manipulation von einzelnen Bits bzw. von Teilworten die auf der CAE 90-40 vorhandenen Meta-Programme MSHIFT, LØDBIT, PUTBIT, MUND, LØDCHA, PUTCHA vorgesehen. Da die Verwendung solcher Funktionen das System zu stark anlagenabhängig macht, wurde auf ihren Einsatz verzichtet, so daß die zur Verfügung stehende FØRTRAN-Bibliothek lediglich für die beiden Hash-Algorithmen die Funktion IEØR, die in Assemblersprache geschrieben werden muß, zu enthalten braucht. Für die Bestimmung einzelner Bits oder Teilworte werden arithmetische Verfahren angewendet, die auf der Multiplikation und ganzzahligen Division mit Zweierpotenzen beruhen. Diese Operationen werden von guten FØRTRAN-Compilern (z.B. Control Data CD3300) durch einfache Shift-Befehle übersetzt und sind daher sehr schnell, und selbst bei schlechten Compilern, die sie als Multiplikation und Division übersetzen, sind sie immer noch schneller als ein Unterprogramm-Sprung. Man erhält die Bits $\geq$ Bit 14 eines Wortes I durch :

$$J = I / 16384 \text{ bzw. } J = I / 16384 * 16384$$

nach rechts verschoben bzw. an ihrer normalen Stelle. Die Bits $<$ Bit 13 eines Wortes I erhält man dagegen durch:

$$J = I - I / 8192 * 8192$$

an ihrer normalen Stelle. Den Wert von Bit 13 eines Wortes erhält man dagegen durch :

$$J = I / 8192 - I / 16384 * 2$$

als 0 oder 1.

5.4 Programme

5.4.1 Triaden-Programme

Die folgenden Triaden-Programme wurden fast unverändert übernommen; lediglich die Area-Abfragen und alle dadurch notwendigen Verzweigungen wurden entfernt :

```
F1_INITIAL(≐F1INIT)
A_LØØKUP(≐ALØØK)
A_ALLØCATE(≐AALLØC)
F2_INITIAL(≐F2INIT)
```

```

GØ_TØ_BØTTØM(≐GØBØTM)
B_RETRIEVAL(≐BRETRI)
C_RETRIEVAL(≐CRETRI)
STØRE_TRIADE(≐STØRE)
DELETE(≐DELETE)
REQUEST_CLASS_I(≐REQCL1)
B_RING_START(≐BRSTRT)
RETRIEVAL_PRØGRAM(≐RETRIV)
F2_RELEASE(≐F2REL)

```

Größere Änderungen waren notwendig bei folgenden Programmen :

```

KEEP_FREE(≐KEEPFR)
PUT_RING_INTØ_ENTITY(≐PUTRIE)
PERM_ØRDER_ØF_REQUEST(≐PEØRDR)

```

Die Routine BACK_TIE_ØN_CØNFLICT wurde erweitert zu einer auf B-, C- und Conflict-Ringen arbeitenden Routine BKTIE. Schließlich wurden die Garbage-Collection-Routinen für F1, HAS und F2 zu der Routine GARCØL zusammengezogen. Die hier genannten Routinen bilden praktisch das Rückgrat des Systems; ohne sie wäre eine Manipulation von Relationen nicht möglich.

5.4.2 Entity-Programme

Die Routine RTRNID wurde ausgehend von der Beschreibung der geplanten Routine RETURN_ID geschrieben, die übrigen Programme des Systems wurden sämtlich neu entwickelt, da zwar die Struktur des Systems beibehalten, die Strukturen von Daten- und Namenspeicher jedoch neu festgelegt wurden.

5.5 Zusammenfassung

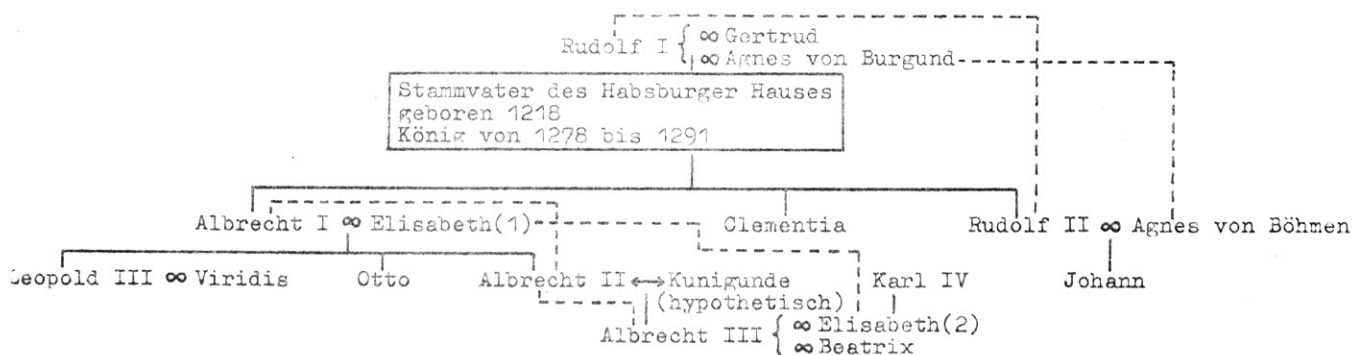
Die vorliegende Implementierung des DATAS-Systems wurde aus dem ursprünglichen Konzept / 6 /, / 11 / entwickelt. Dabei wurde die Struktur des Triadenspeichers weitgehend beibehalten, die von Daten- und Namenspeicher jedoch völlig neu entwickelt. Entsprechendes gilt für diese Speicherbereiche bedienenden Programme. Als wesentlichste Änderung der Systemspezifikationen wurde eine Verringerung der benutzten Wortlänge von 24 auf 16 Bit und damit eine Aufteilung der IDs auf je 2 Worte eingeführt.

Anhang : Beispiele für den Einsatz von DATAS

Die folgenden Beispiele wurden auf der Telefunken TR440 des Rechenzentrums der Universität des Saarlandes gerechnet. Die ersten beiden Beispiele sollen den Einsatz des Dialogsystems demonstrieren. Sie beschreiben die Eingabe eines Stammbaums bzw. einer geometrischen Figur vom Terminal (Bildschirm) aus und ein Abfragen dieser Strukturen. Im dritten Beispiel werden die Unterprogramme der Ebene 1 von DATAS von einem FØRTRAN-Hauptprogramm aus aufgerufen, das im Stapelbetrieb läuft. DATAS wird hier benutzt, um die Struktur eines JK-Master-Slave-Flipflops in einfacher Weise abzuspeichern und auf Rückkopplungen abzusuchen.

A1 Information Retrieval mit dem Dialogsystem : Stammbaum

Der hier behandelte Stammbaum ist ein Teil des Stammbaums des Habsburger Hauses / 12 /. Hier muß angemerkt werden, daß die im Stammbaum angegebene Geliebte Albrechts II., Kunigunde, in dieser Arbeit als hypothetische Größe eingeführt wurde, da in dem als Quelle benutzten Stammbaum Albrecht II. zwar Kinder, aber keine Gemahlin hat.



START PS&BIBTRANS (12.00)

ENDE PS&BIBTRANS (12.00) 0.15
 &STARTE, DIALOG

START DIALOG (1.00)

SYI,1,1,1,30,101,199,

===== BITTE ANWEISUNG EINGEBEN

STS, RUDDLF, STAMMVATER DES HABSBURGER HAUSES,

===== BITTE NAECHSTEN DATENBLOCK EINGEBEN

===== BITTE ANWEISUNG EINGEBEN

STR, VATER, ALBRECHT1, RUDDOLF1, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, VATER, RUDDOLF2, RUDDOLF1, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, VATER, ALBRECHT2, ALBRECHT1, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, VATER, JOHANN, RUDDOLF2, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, VATER, LEOPOLD3, ALBRECHT2, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, VATER, OTTO, ALBRECHT1, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, GEMAHLIN, RUDDOLF2, AGNES(BOE), 2,

===== BITTE ANWEISUNG EINGEBEN

STR, GEMAHLIN, RUDDOLF1, GERTRUD, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, GEMAHLIN, ALBRECHT1, ELISABETH1, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, VATER, CLEMENTIA, RUDDOLF1, 2,

===== BITTE ANWEISUNG EINGEBEN

System-Initialisierung und Eingabe
 des Stammbaumes mit den Relationen

"Vater", "Gemahlin" und "Name" (= "Namensvetter")

STR, VATER, ALBRECHT3, ALBRECHT2, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, VATER, ELISABETH2, KARL4, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, GEMAHLIN, ALBRECHT3, ELISABETH2,

===== BITTE ANWEISUNG EINGEBEN

STR, GEMAHLIN, ALBRECHT3, BEATRIX, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, GEMAHLIN, LEOPOLD3, VIRIDIS, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, GEMAHLIN, RUDDOLF1, AGNES(BU), 2,

===== BITTE ANWEISUNG EINGEBEN

STR, NAME, RUDDOLF2, RUDDOLF1, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, NAME, ALBRECHT2, ALBRECHT1, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, NAME, ALBRECHT3, ALBRECHT1, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, NAME, ELISABETH2, ELISABETH1, 2,

***** FALSCHER ZAHLENEINGABE

===== BITTE ANWEISUNG EINGEBEN

STR, NAME, ELISABETH2, ELISABETH1, 2,

===== BITTE ANWEISUNG EINGEBEN

STR, NAME, AGNES(BOE), AGNES(BU), 2,

===== BITTE ANWEISUNG EINGEBEN

DCS, GATTIN, GEMAHLIN

===== BITTE ANWEISUNG EINGEBEN

FPR,,, RUDDOLF1

***** KEINE ID-INFORMATION

***** KEINE ID-INFORMATION

----- NAME LAUTET

----- RUDDOLF1

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 129

***** BITTE ANWEISUNG EINGEBEN

FDT
 ----- NAME LAUTET
 ----- VATER

***** BITTE ANWEISUNG EINGEBEN

FDH
 ----- NAME LAUTET
 ----- GERTRUD
 TRIADENSEITE = 2 INDEX = 160

***** BITTE ANWEISUNG EINGEBEN

FDN
 ----- NAME LAUTET
 ----- AGNES(BU)
 TRIADENSEITE = 2 INDEX = 225

***** BITTE ANWEISUNG EINGEBEN

FDN
 ----- NAME LAUTET
 ----- NAME
 TRIADENSEITE = 2 INDEX = 231

***** BITTE ANWEISUNG EINGEBEN

CHN,NAMENSVEITER,NAME
 ***** BITTE ANWEISUNG EINGEBEN

FDR,VATER,,RUDOLF1
 ----- NAME LAUTET
 ----- VATER
 ----- NAME LAUTET
 ----- *OALBRECHT
 ----- NAME LAUTET
 ----- RUDOLF1

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 796

***** BITTE ANWEISUNG EINGEBEN

DCS,R1-KINDER,*OALBRECHT
 ***** BITTE ANWEISUNG EINGEBEN

LIE,R1-KINDER,1,
 ----- NAME LAUTET
 ----- ALBRECHT1
 ANZAHL DER ELEMENTE = 3

***** BITTE ANWEISUNG EINGEBEN

LIE,R1-KINDER,2,
 ----- NAME LAUTET

----- RUDOLF2
 ANZAHL DER ELEMENTE = 3

***** BITTE ANWEISUNG EINGEBEN

LIE,R1-KINDER,3,
 ----- NAME LAUTET
 ----- CLEMENTIA
 ANZAHL DER ELEMENTE = 3

***** BITTE ANWEISUNG EINGEBEN

LIE,R1-KINDER,4,
 ***** NUMMER NICHT IM ELEMENTBEREICH

***** BITTE ANWEISUNG EINGEBEN

FDR,VATER,CLEMENTIA,
 ----- NAME LAUTET
 ----- VATER
 ----- NAME LAUTET
 ----- CLEMENTIA
 ----- NAME LAUTET
 ----- RUDOLF1

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 784

***** BITTE ANWEISUNG EINGEBEN

FDR,GEHAHLIN,ALBRECHT3,
 ----- NAME LAUTET
 ----- GATTIN
 ----- NAME LAUTET
 ----- ALBRECHT3
 ----- NAME LAUTET
 ----- *OELISABET

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 820

***** BITTE ANWEISUNG EINGEBEN

DCS,A3-G,*OELISABETH
 ***** BITTE ANWEISUNG EINGEBEN

LIE,A3-G,1,
 ----- NAME LAUTET
 ----- ELISABETH2
 ANZAHL DER ELEMENTE = 2

***** BITTE ANWEISUNG EINGEBEN

LIE,A3-G,2,
 ----- NAME LAUTET
 ----- BLATRIX
 ANZAHL DER ELEMENTE = 2

***** BITTE ANWEISUNG EINGEBEN

Abfragen des Stammbaumes durch Fragen der Klassen 2 und 1 sowie
 durch Fragen über Mengen; Umbenennung der Mengen

```

FDR,,OTTO,ALBRECHT1
----- NAME LAUTET
----- VATER
----- NAME LAUTET
----- OTTO
----- NAME LAUTET
----- ALBRECHT1

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX = 153

===== BITTE ANWEISUNG EINGEBEN

FDR,,RUDOLF2,RUDOLF1
----- NAME LAUTET
----- *OVATER
----- NAME LAUTET
----- RUDOLF2
----- NAME LAUTET
----- RUDOLF1

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX = 838

===== BITTE ANWEISUNG EINGEBEN

LI,*OVATER,1,

***** UNBEKANNTE ANWEISUNG

===== BITTE ANWEISUNG EINGEBEN

LIE,*OVATER,1,
----- NAME LAUTET
----- VATER
ANZAHL DER ELEMENTE = 2

===== BITTE ANWEISUNG EINGEBEN

LIE,*OVATER,2,
----- NAME LAUTET
----- NAMENSVEIT
ANZAHL DER ELEMENTE = 2

===== BITTE ANWEISUNG EINGEBEN

RDE,NAME,1,

***** ENTITY EXISTIERT NICHT

===== BITTE ANWEISUNG EINGEBEN

RDS,NAME,1,

***** ENTITY EXISTIERT NICHT
SYL.ZAHL = 0 SYL.NR. = 1

===== BITTE ANWEISUNG EINGEBEN

FDR,GATTIN,ALBRECHT1,
----- NAME LAUTET
----- GATTIN

```

```

----- NAME LAUTET
----- ALBRECHT1
----- NAME LAUTET
----- ELISABETH1

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX =

===== BITTE ANWEISUNG EINGEBEN

FDR,GEHAHLIN,ALBRECHT2,

***** TRIADE NICHT VORHANDEN
TRIADENSEITE = 2 INDEX =

===== BITTE ANWEISUNG EINGEBEN

FDR,VATER,,ALBRECHT2
----- NAME LAUTET
----- VATER
----- NAME LAUTET
----- *OLEDPOL03
----- NAME LAUTET
----- ALBRECHT2

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX =

===== BITTE ANWEISUNG EINGEBEN

DCS,A3-KINDER,*OLEDPOL03

===== BITTE ANWEISUNG EINGEBEN

FDE,LEOPLO03,A3-KINDER

***** ELEMENT EXISTIERT NICHT

===== BITTE ANWEISUNG EINGEBEN

FDE,LEOPLO03,A3-KINDER
----- ELEMENTBEZIEHUNG ZWISCHEN
----- LEOPLO03 A3-KINDER

===== BITTE ANWEISUNG EINGEBEN

LIE,A3-KINDER,2,
----- NAME LAUTET
----- ALBRECHT3
ANZAHL DER ELEMENTE = 2

===== BITTE ANWEISUNG EINGEBEN

CHD,A3-KINDER,A2-KINDER

***** KEINE ENTITY DIESER NAMENS

===== BITTE ANWEISUNG EINGEBEN

CHD,A2-KINDER,A3-KINDER

===== BITTE ANWEISUNG EINGEBEN

```

Weitere Fragen und Provokation von Fehlermeldungen

STS,KUNIGUNDE,
 ===== BITTE NAECHSTEN DATENBLOCK EINGEBEN
 ===== BITTE NAECHSTEN DATENBLOCK EINGEBEN
 ===== BITTE ANWEISUNG EINGEBEN
 HDRP,,AUS

===== BITTE ANWEISUNG EINGEBEN
 STR,GELIEBTE,ALBRECHT2,KUNIGUNDE,2,

===== BITTE ANWEISUNG EINGEBEN

DCS,MAETRESSE,GELIEBTE

===== BITTE ANWEISUNG EINGEBEN

DCS,KEBSWEIB,MAETRESSE

===== BITTE ANWEISUNG EINGEBEN

FDR,GATTIN,,
 ----- NAME LAUTET
 ----- GATTIN

***** KEINE ID-INFORMATION

***** KEINE ID-INFORMATION

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 748

===== BITTE ANWEISUNG EINGEBEN

FDT
 ----- NAME LAUTET
 ----- RUDOLF2

===== BITTE ANWEISUNG EINGEBEN

FDR,GEHAHLIN,RUDOLF2,
 ----- NAME LAUTET
 ----- GATTIN
 ----- NAME LAUTET
 ----- RUDOLF2
 ----- NAME LAUTET
 ----- AGNES(BOE)

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 748

===== BITTE ANWEISUNG EINGEBEN

FDN,2,838,

***** BOTTOM SCHON ERREICHT
 TRIADENSEITE = 2 INDEX = 838

===== BITTE ANWEISUNG EINGEBEN

FDR,GATTIN,,
 ----- NAME LAUTET
 ----- GATTIN

***** KEINE ID-INFORMATION

***** KEINE ID-INFORMATION

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 748

===== BITTE ANWEISUNG EINGEBEN

FDN
 ----- NAME LAUTET
 ----- RUDOLF1
 TRIADENSEITE = 2 INDEX = 147

===== BITTE ANWEISUNG EINGEBEN

FDR,GATTIN,RUDOLF1,
 ----- NAME LAUTET
 ----- GATTIN
 ----- NAME LAUTET
 ----- RUDOLF1
 ----- NAME LAUTET
 ----- *OGERTRUD

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 832

===== BITTE ANWEISUNG EINGEBEN

RDE,*OGERTRUD,1,
 DATENLAENGE = 10 KLASSE = 3

1 2 B 2 J

===== BITTE ANWEISUNG EINGEBEN

FDR,,ALBRECHT2,

***** KEINE ID-INFORMATION
 ----- NAME LAUTET
 ----- ALBRECHT2

***** KEINE ID-INFORMATION

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 123

===== BITTE ANWEISUNG EINGEBEN

FDT
 ----- NAME LAUTET
 ----- ALBRECHT1

===== BITTE ANWEISUNG EINGEBEN

FDN

Erweitern der Struktur; weitere Fragen; Lesen der Daten einer
 Multi-Value-Entity

```

----- NAME LAUTET
----- VATER
TRIADENSEITE = 2 INDEX = 736

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

FDR
----- NAME LAUTET
----- KUNIGUNDE
TRIADENSEITE = 2 INDEX = 907

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

FDR
***** BOTTOM SCHON ERREICHT
TRIADENSEITE = 2 INDEX = 907

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

FDR,GELIEBTE,,KUNIGUNDE
----- NAME LAUTET
----- KEBSWEIB
----- NAME LAUTET
----- ALBRECHT2
----- NAME LAUTET
----- KUNIGUNDE

```

```

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX = 913

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

RDE,KUNIGUNDE,1,
DATENLAENGE = 242 KLASSE = 2

```

```

BE ALBRECHT2 ZWAR KINDER, ABER
IN HAT, ENTSTEHT HIER EINE
DE SITUATION, DIE DURCH EINFUEH-
YPOTHETISCHEN GELIEBTEN NAMENS
LWEST WIRD.

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

RDS,GELIEBTE,1,
----- NAME LAUTET
----- KEBSWEIB
SYN.ZAHL = 3 SYN.NR. = 1

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

RDS,GELIEBTE,2,
----- NAME LAUTET
----- MAETRESSE
SYN.ZAHL = 3 SYN.NR. = 2

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

RDS,MAETRESSE,3,
----- NAME LAUTET
----- GELIEBTE
SYN.ZAHL = 3 SYN.NR. = 3

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

FDR,,,VIRIDIS

```

```

***** KEINE ID-INFORMATION

```

```

***** KEINE ID-INFORMATION

```

```

----- NAME LAUTET
----- VIRIDIS

```

```

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX =

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

FDT
----- NAME LAUTET
----- GATTIN

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

FDR

```

```

***** BOTTOM SCHON ERREICHT
TRIADENSEITE = 2 INDEX =

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

FDR,GATTIN,,VIRIDIS
----- NAME LAUTET
----- GATTIN
----- NAME LAUTET
----- LEOPOLD3
----- NAME LAUTET
----- VIRIDIS

```

```

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX =

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

FDR,,,KARL4,ELISABETH2,2,

```

```

***** TRIADE NICHT VORHANDEN
TRIADENSEITE = 2 INDEX =

```

```

===== BITTE ANWEISUNG EINGEBEN

```

```

FDR,,,ELISABETH2,KARL4

```

```

***** KEINE ID-INFORMATION

```

```

***** KEINE ID-INFORMATION

```

```

----- NAME LAUTET
----- ELISABETH2

```

Abfragen der neuen Teile der Struktur; Lesen von Daten und Synonymen

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 811

===== BITTE ANWEISUNG EINGEBEN

FDR,,ELISABETH2,KARL4
 ----- NAME LAUTET
 ----- VATER
 ----- NAME LAUTET
 ----- ELISABETH2
 ----- NAME LAUTET
 ----- KARL4

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 811

===== BITTE ANWEISUNG EINGEBEN

FDT
 ----- NAME LAUTET
 ----- KARL4

===== BITTE ANWEISUNG EINGEBEN

FDN
 ----- NAME LAUTET
 ----- GATTIN
 TRIADENSEITE = 2 INDEX = 249

===== BITTE ANWEISUNG EINGEBEN

FDN
 ----- NAME LAUTET
 ----- ELISABETH1
 TRIADENSEITE = 2 INDEX = 261

===== BITTE ANWEISUNG EINGEBEN

FDN
 ***** BOTTOM SCHON ERREICHT
 TRIADENSEITE = 2 INDEX = 261

===== BITTE ANWEISUNG EINGEBEN

SYN
 STOP

ENDE DIALOG (1.00) 7.59

Absuchen eines C-Ringes und
 Beenden des Dialogs; die ins-
 gesamt benötigte Rechenzeit
 betrug 7.59 sec.

DATUM: 21.01.74
 ANF.-ZEIT: 12.55 UHR
 END-ZEIT: 13.57 UHR
 BEN: 4101101003**BKDETEST
 FKZ: WECK
 KSB: 27 K
 TSB: 50 K
 PSB: 70 K
 ORS: 30 S
 KZS: 60 SEC
 RK-ZEIT: 8.732 SEC
 TRANSP.INDEX: 7.666 SEC

MINIMALWERTE DES ABSCHNITTES:

KSB: 27 K
 TSB: 20 K
 PSB: 43 K

Liste der benötigten Betriebs-
 mittel auf einer TR 440:

KSB: Kernspeicher-Bereich
 TSB: Trommelspeicher-Bereich
 PSB: Plattenspeicher-Bereich

Die Wartezeiten am Display
 können sehr unterschiedlich
 sein; sie sind abhängig vom
 Betriebssystem der Maschine, von
 der Anzahl der gerade laufenden
 Jobs und von der Menge der im
 Augenblick zur Verfügung stehen-
 den Betriebsmittel. Die an der
 hier benützten TR 440 gemachten
 Erfahrungen zeigten, daß die
 Antworten auf DATAS-Befehle prak-
 tisch ebenso schnell erfolgen wie
 auf Programmiersystem-Kommandos
 wie zum Beispiel Ein- oder Aus-
 schalten des Druckerprotokolls.
 Je nach Belastung der Maschine
 schwankten diese Zeiten von einer
 Sekunde bis zu mehreren Minuten!

A2 Einsatz des Dialogsystems zur Bearbeitung graphischer Daten

START PS&BI&TRANS (12,00)

ENDE PS&BI&TRANS (12,00) 0.15
 ▣STARTE,DIALOG

START DIALOG (1,00)

SYI,1,1,1,30,101,199,

***** BITTE ANWEISUNG EINGEBEN

STR,WINKEL,ECKE EINES POLYGONS,

***** BITTE NAECHSTEN DATENBLOCK EINGEBEN

***** BITTE NAECHSTEN DATENBLOCK EINGEBEN

***** BITTE ANWEISUNG EINGEBEN

STR,WINKEL,FIG1,P1,2,

***** BITTE ANWEISUNG EINGEBEN

STR,WINKEL,FIG1,P2,2,

***** BITTE ANWEISUNG EINGEBEN

STR,WINKEL,FIG1,P3,2,

***** BITTE ANWEISUNG EINGEBEN

STR,WINKEL,FIG1,P4,2,

***** BITTE ANWEISUNG EINGEBEN

STR,WINKEL,FIG2,P3,2,

***** BITTE ANWEISUNG EINGEBEN

STR,WINKEL,FIG2,P4,2,

***** BITTE ANWEISUNG EINGEBEN

STR,WINKEL,FIG2,P5,2,

***** BITTE ANWEISUNG EINGEBEN

STR,TYP,FIG1,RECHTECK,2,

***** BITTE ANWEISUNG EINGEBEN

STR,TYP,FIG2,DREIECK,2,

***** BITTE ANWEISUNG EINGEBEN

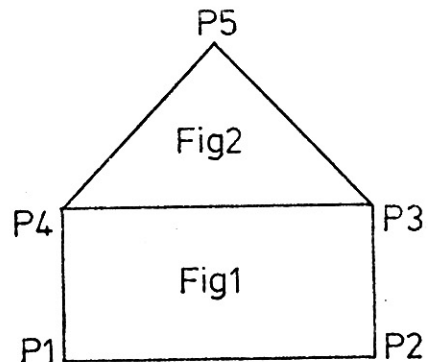
FDK,TYP,,

----- NAME LAUTET

System-Initialisierung

Abspeichern des Begriffs
 "Winkel" als Single-
 Value-Entity

Abspeichern der
 untenstehenden
 Figur durch die
 Relationen "Winkel"
 und "Typ"



Beginn Abfragen der Fig

```

----- TYP
***** KEINE ID-INFORMATION
***** KEINE ID-INFORMATION
***** TRIADE GEFUNDEN
  TRIADENSEITE =    2  INDEX =    763
===== BITTE ANWEISUNG EINGEBEN

  FDT
----- NAME LAUTET
----- FIG1

===== BITTE ANWEISUNG EINGEBEN

  FDR,TYP,FIG1,
----- NAME LAUTET
----- TYP
----- NAME LAUTET
----- FIG1
----- NAME LAUTET
----- RECHTECK

***** TRIADE GEFUNDEN
  TRIADENSEITE =    2  INDEX =    763
===== BITTE ANWEISUNG EINGEBEN

  FDN
----- NAME LAUTET
----- FIG2
  TRIADENSEITE =    2  INDEX =    171
===== BITTE ANWEISUNG EINGEBEN

  FDR,TYP,FIG2,
----- NAME LAUTET
----- TYP
----- NAME LAUTET
----- FIG2
----- NAME LAUTET
----- DREIECK

***** TRIADE GEFUNDEN
  TRIADENSEITE =    2  INDEX =    171
===== BITTE ANWEISUNG EINGEBEN

  FDN
***** BOTTOM SCHON ERREICHT
  TRIADENSEITE =    2  INDEX =    171
===== BITTE ANWEISUNG EINGEBEN

  FDR,WINKEL,,
----- NAME LAUTET
----- WINKEL

***** KEINE ID-INFORMATION

```

```

***** KEINE ID-INFORMATION
***** TRIADE GEFUNDEN
  TRIADENSEITE =    2  INDEX =    109
===== BITTE ANWEISUNG EINGEBEN

  FDT
----- NAME LAUTET
----- FIG1

===== BITTE ANWEISUNG EINGEBEN

  FDR,WINKEL,FIG1,
----- NAME LAUTET
----- WINKEL
----- NAME LAUTET
----- FIG1
----- NAME LAUTET
----- *OP1

***** TRIADE GEFUNDEN
  TRIADENSEITE =    2  INDEX =    709
===== BITTE ANWEISUNG EINGEBEN

  DCS,R-PUNKTE,*OP1

===== BITTE ANWEISUNG EINGEBEN

  LIE,R-PUNKTE,1,
----- NAME LAUTET
----- P1
  ANZAHL DER ELEMENTE =    4

===== BITTE ANWEISUNG EINGEBEN

  LIE,R-PUNKTE,2,
----- NAME LAUTET
----- P2
  ANZAHL DER ELEMENTE =    4

===== BITTE ANWEISUNG EINGEBEN

  LIE,R-PUNKTE,3,
----- NAME LAUTET
----- P3
  ANZAHL DER ELEMENTE =    4

===== BITTE ANWEISUNG EINGEBEN

  LIE,R-PUNKTE,4,
----- NAME LAUTET
----- P4
  ANZAHL DER ELEMENTE =    4

===== BITTE ANWEISUNG EINGEBEN

  LIE,R-PUNKTE,5,

***** NUMMER NICHT IM ELEMENTBEREICH

```

Abfragen der Figur durch Fragen der Klassen 2 und 1 und Fragen
über die dabei erzeugten Mengen

===== BITTE ANWEISUNG EINGEBEN

FDE,P5,R-PUNKTE
 ----- KEINE ELEMENTBEZIEHUNG
 ----- P5 R-PUNKTE

===== BITTE ANWEISUNG EINGEBEN
 DRPROTOKOLL,AUS

FDR,WINKEL,FIG1,
 ----- NAME LAUTET
 ----- WINKEL
 ----- NAME LAUTET
 ----- FIG1
 ----- NAME LAUTET
 ----- R-PUNKTE

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 709

===== BITTE ANWEISUNG EINGEBEN

FDT,2,105,
 ----- NAME LAUTET
 ----- FIG1

===== BITTE ANWEISUNG EINGEBEN

FDN
 ----- NAME LAUTET
 ----- FIG2
 TRIADENSEITE = 2 INDEX = 724

===== BITTE ANWEISUNG EINGEBEN

FDR,WINKEL,FIG2,
 ----- NAME LAUTET
 ----- WINKEL
 ----- NAME LAUTET
 ----- FIG2
 ----- NAME LAUTET
 ----- *OP3

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 760

===== BITTE ANWEISUNG EINGEBEN

CHN,D-PUNKTE,*OP3

===== BITTE ANWEISUNG EINGEBEN

LIF,D-PUNKTE,1,
 ----- NAME LAUTET
 ----- P3
 ANZAHL DER ELEMENTE = 3

===== BITTE ANWEISUNG EINGEBEN

LIE,D-PUNKTE,2,

----- NAME LAUTET
 ----- P4
 ANZAHL DER ELEMENTE = 3

===== BITTE ANWEISUNG EINGEBEN

LIF,D-PUNKTE,3,
 ----- NAME LAUTET
 ----- P5
 ANZAHL DER ELEMENTE = 3

===== BITTE ANWEISUNG EINGEBEN

FDR,WINKEL,FIG2,
 ----- NAME LAUTET
 ----- WINKEL
 ----- NAME LAUTET
 ----- FIG2
 ----- NAME LAUTET
 ----- *OP3

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 760

===== BITTE ANWEISUNG EINGEBEN

DCS,D-PUNKTE,*OP3
 ----- UEBERSPEICHERN VON
 ----- D-PUNKTE
 ++++ JA

===== BITTE ANWEISUNG EINGEBEN

FDR,WINKEL,FIG2,
 ----- NAME LAUTET
 ----- WINKEL
 ----- NAME LAUTET
 ----- FIG2
 ----- NAME LAUTET
 ----- D-PUNKTE

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 760

===== BITTE ANWEISUNG EINGEBEN

FDR,WINKEL,,P3
 ----- NAME LAUTET
 ----- WINKEL
 ----- NAME LAUTET
 ----- *OFIG1
 ----- NAME LAUTET
 ----- P3

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 739

===== BITTE ANWEISUNG EINGEBEN

DCS,FIGUREN,*OFIG1

===== BITTE ANWEISUNG EINGEBEN

Weitere Fragen und Umbenennungen von Entities; Demonstration des Programmiersystem-Kommandos "DRPROTOKOLL", das den Dialog unterbricht, bis er durch einen weiteren Anruf ans Programmiersystem wieder aufgenommen wird

```

FDR,WINKEL,,P4
----- NAME LAUTET
----- WINKEL
----- NAME LAUTET
----- FIGUREN
----- NAME LAUTET
----- P4

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX = 757

***** BITTE ANWEISUNG EINGEBEN

FDE,FIG1,FIGUREN
----- ELEMENTBEZIEHUNG ZWISCHEN
----- FIG1 FIGUREN

***** BITTE ANWEISUNG EINGEBEN

LIE,FIGUREN,2,
----- NAME LAUTET
----- FIG2
ANZAHL DER ELEMENTE = 2

***** BITTE ANWEISUNG EINGEBEN

LIE,FIG1,1,
***** KEINE MENGE

***** BITTE ANWEISUNG EINGEBEN

LIE,FIG3,1,
***** MENGE EXISTIERT NICHT

***** BITTE ANWEISUNG EINGEBEN

FDE,FIG3,FIGUREN
***** ELEMENT EXISTIERT NICHT

***** BITTE ANWEISUNG EINGEBEN

RDE,WINKEL,1,
DATENLAENGE = 98 KLASSE = 1

5 LECHE EINES POLYGONS
LEICHZEITIG MEHREREN POLYGONEN
EREN.

***** BITTE ANWEISUNG EINGEBEN

XFER(LGU,,PUN)

***** ZEICHEN NICHT IDENTIFIZIERBAR

***** BITTE ANWEISUNG EINGEBEN

```

```

MIST

***** UNBEKANNTE ANWEISUNG

***** BITTE ANWEISUNG EINGEBEN

FDR,,P4

***** KEINE ID-INFORMATION

***** KEINE ID-INFORMATION
----- NAME LAUTET
----- P4

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX = 718

***** BITTE ANWEISUNG EINGEBEN

FDT
----- NAME LAUTET
----- WINKEL

***** BITTE ANWEISUNG EINGEBEN

FDN

***** BOTTOM SCHON ERREICHT
TRIADENSEITE = 2 INDEX = 718

***** BITTE ANWEISUNG EINGEBEN

FDR,,FIG1,

***** KEINE ID-INFORMATION
----- NAME LAUTET
----- FIG1

***** KEINE ID-INFORMATION

***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX = 99

***** BITTE ANWEISUNG EINGEBEN

FDT
----- NAME LAUTET
----- P1
KANN G -----

ANGEHO ***** BITTE ANWEISUNG EINGEBEN

FDN
----- NAME LAUTET
----- P2
TRIADENSEITE = 2 INDEX = 135

***** BITTE ANWEISUNG EINGEBEN

FDN
----- NAME LAUTET
----- P3
TRIADENSEITE = 2 INDEX = 129

```

Lesen von Daten und Provokation von Fehlermeldungen

===== BITTE ANWEISUNG EINGEBEN

FDN
 ----- NAME LAUTET
 ----- P4
 TRIADENSEITE = 2 INDEX = 712

===== BITTE ANWEISUNG EINGEBEN

FDN
 ----- NAME LAUTET
 ----- RECHTECK
 TRIADENSEITE = 2 INDEX = 153

===== BITTE ANWEISUNG EINGEBEN

FDR,,FIG1,RECHTECK
 ----- NAME LAUTET
 ----- TYP
 ----- NAME LAUTET
 ----- FIG1
 ----- NAME LAUTET
 ----- RECHTECK

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 153

===== BITTE ANWEISUNG EINGEBEN

RDS,FIGUREN,1,
 ----- NAME LAUTET
 ----- FIGUREN
 SYN,ZAHL = 2 SYN,NR. = 1

===== BITTE ANWEISUNG EINGEBEN

RDS,FIGUREN,2,
 ----- NAME LAUTET
 ----- *OFIG1
 SYN,ZAHL = 2 SYN,NR. = 2

===== BITTE ANWEISUNG EINGEBEN

RDS,*OFIG1,1,
 ----- NAME LAUTET
 ----- FIGUREN
 SYN,ZAHL = 2 SYN,NR. = 1

===== BITTE ANWEISUNG EINGEBEN

RDS,*OFIG1,2,
 ----- NAME LAUTET
 ----- *OFIG1
 SYN,ZAHL = 2 SYN,NR. = 2

===== BITTE ANWEISUNG EINGEBEN

DLN,FIGUREN

===== BITTE ANWEISUNG EINGEBEN

DLN,*OFIG1

***** ILLEGALER LOESCHVERSUCH

===== BITTE ANWEISUNG EINGEBEN

STE, FIG3,*OFIG1

***** ILLEGALER NAME

===== BITTE ANWEISUNG EINGEBEN

LIE,*OFIG1,1,

***** KEINE ID-INFORMATION

===== BITTE ANWEISUNG EINGEBEN

LIE,*OFIG1,2,
 ----- NAME LAUTET
 ----- FIG1
 ANZAHL DER ELEMENTE = 3

===== BITTE ANWEISUNG EINGEBEN
 HDRPROTOKOLL,AUS

FDR,WINKEL,,P3
 ----- NAME LAUTET
 ----- WINKEL
 ----- NAME LAUTET
 ----- *1FIG1
 ----- NAME LAUTET
 ----- P3

***** TRIADE GEFUNDEN
 TRIADENSEITE = 2 INDEX = 739

===== BITTE ANWEISUNG EINGEBEN

CHN,*OFIG1,*1FIG1
 ----- UEBERSPEICHERN VON
 ----- *OFIG1
 ++++ OK

===== BITTE ANWEISUNG EINGEBEN

LIE,*OFIG1,1,
 ----- NAME LAUTET
 ----- FIG1
 ANZAHL DER ELEMENTE = 2

===== BITTE ANWEISUNG EINGEBEN

STR,,,,,

***** FALSCHER ZAHLENEINGABE

===== BITTE ANWEISUNG EINGEBEN

STR,,,,0,

Demonstration von Synonymen und Provokation weiterer Fehler-
 meldungen

```

***** ILLEGALER NAME
***** ILLEGALER NAME
***** ILLEGALER NAME
***** ILLEGALER RELATIONS-NAME
***** BITTE ANWEISUNG EINGEBEN
    STR,
***** FALSCHER PARAMETERANZAHL
***** BITTE ANWEISUNG EINGEBEN
    ON,

```

Demonstration des Programmiersystem-Kommandos "N.", das den Dialog unterbricht und einen Seitenvorschub auf dem Drucker erzeugt; der Dialog wird nach einem weiteren Anruf ans Programmiersystem wieder aufgenommen.

Löschen einer Triade und Absuchen des C-Ringes, der sie enthielt; man beachte, daß jetzt am Platz von P2 eine ungültige (gelöschte) Triade steht.

```

DLR,WINKEL,FIG1,P2,0,
***** BITTE ANWEISUNG EINGEBEN

    FOR,WINKEL,FIG1,
----- NAME LAUTET
----- WINKEL
----- NAME LAUTET
----- FIG1
----- NAME LAUTET
----- *1P1
***** TRIADE GEFUNDEN
    TRIADENSEITE = 2 INDEX = 709
***** BITTE ANWEISUNG EINGEBEN

    LIE,*1P1,2,
----- NAME LAUTET
----- P3
    ANZAHL DER ELEMENTE = 3
***** BITTE ANWEISUNG EINGEBEN

    FOR,,FIG1,
***** KEINE ID-INFORMATION
----- NAME LAUTET
----- FIG1
***** KEINE ID-INFORMATION
***** TRIADE GEFUNDEN
    TRIADENSEITE = 2 INDEX = 99
***** BITTE ANWEISUNG EINGEBEN

    FDT
----- NAME LAUTET
----- P1
***** BITTE ANWEISUNG EINGEBEN

    FDN
***** KEINE ID-INFORMATION
    TRIADENSEITE = 2 INDEX = 135
***** BITTE ANWEISUNG EINGEBEN

    FDN
----- NAME LAUTET
----- P3
    TRIADENSEITE = 2 INDEX = 129
***** BITTE ANWEISUNG EINGEBEN

    FDN
----- NAME LAUTET
----- P4

```

```

TRIADENSEITE = 2 INDEX = 712
===== BITTE ANWEISUNG EINGEBEN
FON
----- NAME LAUTET
----- RECHTECK
TRIADENSEITE = 2 INDEX = 153
===== BITTE ANWEISUNG EINGEBEN
FON
***** BOTTOM SCHON ERREICHT
TRIADENSEITE = 2 INDEX = 153
===== BITTE ANWEISUNG EINGEBEN
SYG
===== BITTE ANWEISUNG EINGEBEN
FDR,,FIG1,
***** KEINE ID-INFORMATION
----- NAME LAUTET
----- FIG1
***** KEINE ID-INFORMATION
***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX = 99
===== BITTE ANWEISUNG EINGEBEN
FOT
----- NAME LAUTET
----- P1
===== BITTE ANWEISUNG EINGEBEN
FON
----- NAME LAUTET
----- P3
TRIADENSEITE = 2 INDEX = 129
===== BITTE ANWEISUNG EINGEBEN
FON
----- NAME LAUTET
----- P4
TRIADENSEITE = 2 INDEX = 123
===== BITTE ANWEISUNG EINGEBEN
FON
----- NAME LAUTET
----- RECHTECK
TRIADENSEITE = 2 INDEX = 153
===== BITTE ANWEISUNG EINGEBEN

```

```

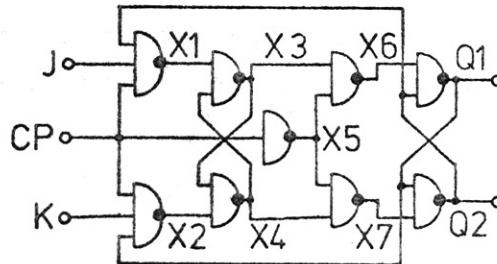
FON
***** BOTTOM SCHON ERREICHT
TRIADENSEITE = 2 INDEX = 153
===== BITTE ANWEISUNG EINGEBEN
FDR,WINKEL,FIG1,
----- NAME LAUTET
----- WINKEL
----- NAME LAUTET
----- FIG1
----- NAME LAUTET
----- *1P1
***** TRIADE GEFUNDEN
TRIADENSEITE = 2 INDEX = 709
===== BITTE ANWEISUNG EINGEBEN
SYD
STOP

```

Demonstration der Garbage-Collection im Triadenspeicher:
 Die ungültige Triade ist verschwunden, und P4 hat einen neuen Index erhalten, da hier ein Conflict-Ring aufgelöst wurde.

A3 Einsatz von DATAS im Stapelbetrieb : Rückkopplungen in einem JK-Flipflop

Zunächst wird, nach der System-Initialisierung, die Struktur der Schaltung des folgenden JK-Master-Slave-Flipflops eingegeben :



Dies geschieht durch wiederholten Aufruf von STRELE. Diese Aufrufe wurden absichtlich nicht in Form einer Laufanweisung geschrieben, da in diesem Falle die Parameter für jeden Aufruf von Karten eingelesen und umcodiert werden müßten, was einigen Zeitverlust bedeuten würde. Da für jede Verknüpfung sowieso ein eigenes Tripel von Parametern angegeben werden müßte, ist der Aufwand beim einzelnen Ablocken der Aufrufe nicht höher als beim Aufruf in einer Laufanweisung. Lohnend wird das Abspeichern einer Struktur in einem System von Laufanweisungen erst, wenn die Struktur bestimmte Regelmäßigkeiten größeren Umfanges besitzt, was hier nicht der Fall ist. Da man den Inverter der oben angegebenen Schaltung auch als NAND-Gatter mit nur einem Eingang auffassen kann, wurde die Struktur durch eine einzige Relation

NAND(Ausgang,Eingang)

dargestellt.

Ausgehend von Q1 werden alle Punkte der Schaltung bestimmt, die sich vom Ausgangspunkt aus über genau 1,2,3,...,9 Gatter erreichen lassen. Die Punkte jeder dieser "Generationen" werden in Multi-Value-Entities M1 bis M9 als Elemente gespeichert. Diese Mengen werden der Reihe nach aufgelistet. Kommt in einer Menge Mn der Punkt Q1 vor, so liegt Q1 in einer Rückkopplung über genau n Gatter.

Das hier angegebene einfache Programm läßt sich ohne Schwierigkeiten so abändern, daß es beliebige Schaltwerke vom Kartenleser aus einliest und die Zugehörigkeit beliebiger Punkte zu Rückkopplungen beliebiger maximaler Länge überprüft. Darauf wurde hier jedoch verzichtet, um das Beispiel möglichst einfach und leicht verständlich zu machen. Dieses Programm läßt sich auch ohne weiteres benutzen, um in einem Stammbaum wie in A1 die vorkommenden Personen nach Generationen zu sortieren; allerdings scheint dann die Frage nach Rückkopplung nicht mehr besonders sinnvoll.

```

PROGRAM COMPLEX RETRIEVAL
COMMON/7/IP(2048)/INAME/IN(2048)/IENT1/IE(2048)
COMMON/DI/PAGTAB/BUFFER/IBUF(80),LBUF
INTEGER NAME(5,22),NJ(5),NCP(5),NK(5),N1(5),N2(5),N3(5),N4(5),N5(5),
N6(5),N7(5),NQ1(5),NQ2(5),NAND(5),NX(5,9),NA(5),NB(5),NC(5),ND(5),
NE(5)
INTEGER PAGTAB(109),IFELD(40),IKT(25),NR(9),IDATA(1)
EQUIVALENCE(NAME(1,1),NJ(1)),(NAME(1,2),NCP(1)),(NAME(1,3),NK(1))
EQUIVALENCE(NAME(1,4),N1(1)),(NAME(1,5),N2(1)),(NAME(1,6),N3(1)),(
INAME(1,7),N4(1)),(NAME(1,8),N5(1)),(NAME(1,9),N6(1)),(NAME(1,10),N
27(1)),(NAME(1,11),NQ1(1)),(NAME(1,12),NQ2(1)),(NAME(1,13),NAND(1))
EQUIVALENCE(NAME(1,14),NX(1,1))
DATA INL,IPL,IEL,NF1,NH,NNC/3*1,30,101,199/,IMAX/9/
LBUF=80
DO 1000 I=1,22
READ(5,1001) IBUF
CALL DINCDD(IFELD,IKT)
DO 1000 J=1,5
1000 NAME(J,I)=IFELD(J)
EINLESEN UND UMCODIEREN DER NAMEN
CALL SYINT(INL,IPL,IEL,NF1,NH,NNC,LBUF)
SYSTEM-INITIALISIERUNG
CALL STRELE(NAND,N1,NJ,2)
CALL STRELE(NAND,N1,NCP,2)
CALL STRELE(NAND,N1,NQ2,2)
CALL STRELE(NAND,N2,NK,2)
CALL STRELE(NAND,N2,NCP,2)
CALL STRELE(NAND,N2,NQ1,2)
CALL STRELE(NAND,N3,N4,2)
CALL STRELE(NAND,N3,N1,2)
CALL STRELE(NAND,N4,N2,2)
CALL STRELE(NAND,N4,N3,2)
CALL STRELE(NAND,N5,NCP,2)
CALL STRELE(NAND,N6,N3,2)
CALL STRELE(NAND,N6,N5,2)
CALL STRELE(NAND,N7,N4,2)
CALL STRELE(NAND,N7,N5,2)
CALL STRELE(NAND,NQ1,N6,2)
CALL STRELE(NAND,NQ1,NQ2,2)
CALL STRELE(NAND,NQ2,N7,2)
CALL STRELE(NAND,NQ2,NQ1,2)
ABSPEICHERN DER STRUKTUR
BEGINN SUCHEN ALLER DIREKTEN EINGAENGE VON Q1
NA(1)=0
FORELE FRAGT NUR DIE BEIDEN ERSTEN CHARACTER AB
CALL FORELE(NAND,NA,NQ1,IPACET,IND,ICOD)
NR(1)=1
VORBESETZEN DER ELEMENTANZAHL
DO 1002 I=1,5
1002 NB(I)=IKX(I,1)
UEBERTRAGEN DES NAMENS
IF(ICOD.EQ.3) GO TO 1003
ABFRAGE, OB TERMINAL ODER C-RING
BEGINN TERMINAL-BEHANDLUNG
CALL STELEM(NA,1,10ATA,NB)
ERZEUGEN EINER EINELEMENTIGEN MENGE
GO TO 1004
BEGINN C-RING-BEHANDLUNG
1003 CALL LIELEM(NC,NA,NR(1),(ID1,ID2)
BESTIMMEN DER ELEMENTANZAHL
CALL CHNAME(NB,NA)

```

```

C      NEUBENENNUNG DER MENGE
C      BEGINN SUCHEN ALLER INDIKREKTEN VURGAENGER VON Q1
1004 I=2
C      BEGINN NAECHSTE GENERATION
1005 J=1
      DO 1006 II=1,5
1006 NR(II)=NR(II,I)
C      UEBERTRAGEN DES NAMENS
C      BEGINN NAECHSTES ELEMENT DER GENERATION
1007 NR=J
      CALL LIELEM(NC,NB,NM,ID1,ID2)
C      BESTIMMUNG DES J-TEN ELEMENTS
C      NR(OUT)=ANZAHL DER ELEMENTE
      NA(1)=0
C      ES SOLL NACH NA GESUCHT WERDEN
      CALL FDELEM(NAND,NA,NC,IPAGE1,IND,ICOD)
      NR(I)=1
C      VORBESEITZEN DER ELEMENTANZAHL
      IF(ICOD.LE.0) GO TO 1008
C      KEINE TRIADE GEFUNDEN: EINGANG
      IF(ICOD.EQ.3) GO TO 1009
C      ABFRAGE, OB TERMINAL ODER C-RING
C      BEGINN TERMINAL-BEHANDLUNG
      CALL FDELEM(NA,ND,ICOD)
C      SUCHEN DES ELEMENTS IN DER GRUNDMENGE DER GENERATION
      IF(ICOD.NE.0) GO TO 1008
C      BEREITS VORHANDENE ELEMENTE BRAUCHEN NICHT MEHR ABGESPEICHERT
C      WERDEN
      CALL STELEM(NA,1,IData,ND)
C      ABSPEICHERN ALS ELEMENT
      GO TO 1008
C      BEGINN C-RING-BEHANDLUNG
1009 K=1
C      BEGINN NAECHSTES ELEMENT DES C-RINGES
1010 NR=K
      CALL LIELEM(NC,NA,NN,ID1,ID2)
C      BESTIMMUNG DES K-TEN ELEMENTS
C      NR(OUT)=ANZAHL DER ELEMENTE
      CALL FDELEM(NC,ND,ICOD)
C      SUCHEN DES ELEMENTS IN DER GRUNDMENGE DER GENERATION
      IF(ICOD.NE.0) GO TO 1011
C      BEREITS VORHANDENE ELEMENTE BRAUCHEN NICHT MEHR ABGESPEICHERT
C      WERDEN
      CALL STELEM(NC,1,IData,ND)
C      ABSPEICHERN ALS ELEMENT DER GRUNDMENGE
1011 K=K+1
C      NAECHSTES ELEMENT DES C-RINGES
      IF(K.LE.NN) GO TO 1010
C      ALLE ELEMENTE DES C-RINGES ABGEARBEITET
      CALL DLSNGL(NA)
C      LUESCHEN DER ALTEN MENGE
1008 J=J+1
C      NAECHSTES ELEMENT DER VORIGEN GENERATION
      IF(J.LE.NM) GO TO 1007
C      ALLE ELEMENTE DIESER GENERATION ABGEARBEITET
      CALL LIELEM(NC,ND,NR(I),ID1,ID2)
C      BESTIMMUNG DER ELEMENTANZAHL DER NEULN GENERATION
      I=I+1
C      NAECHSTE GENERATION
      IF(K.EQ.0) GO TO 1012
C      LEBERE MENGE, VURZEITIGER ABBRUCH
      DO 1013 J=1,5

```

```

1013 NB(J)=ND(J)
C   AUSTAUSCHEN DES NAMENS
    IF(I.LE.IMAX) GO TO 1005
C   ALLE GEWÜNSCHTEN GENERATIONEN ABGEARBEITET
    GO TO 1014
1012 WRITE(6,1015)
C   FEHLERMELDUNG BEI VORZEITIGEM ABBRUCH
1014 WRITE(6,1016)
C   UEBERSCHRIFT, BEGINN TABELLE DER GENERATIONEN
    I=1-1
    DO 1017 J=1,I
        WRITE(6,1018) J,NR(J)
C   GENERATIONSNUMMER UND ELEMENTANZAHL
    DO 1019 K=1,5
1019 NA(K)=NX(K,J)
C   UEBERTRAGEN DES NAMENS
    K=NR(J)
C   ANZAHL DER ELEMENTE DER GENERATION
    DO 1020 II=1,K
        JJ=II
1020 CALL LIELEH(NB,NA,JJ,ID1,ID2)
C   AUFLISTEN DER ELEMENTE
        CALL FDELEM(NQ1,NA,ICDD)
C   SUCHEN NACH RUECKKOPPLUNGEN
        IF(ICDD.NE.0) WRITE(6,1021)
C   RUECKKOPPLUNG GEFUNDEN
1017 CONTINUE
1001 FORMAT(80A1)
1015 FORMAT(1H1////6X,19HVORZEITIGER ABBRUCH)
1016 FORMAT(1H1////6X,22HLISTE DER GENERATIONEN)
1018 FORMAT(1H0/4X,14HGENERATION NR.,13//4X,14HZAHL DER EL. =,13)
1021 FORMAT(1H0,19(1H*)/4X,13HRUECKKOPPLUNG/1X,19(1H*))
    END

```

ERZEUGTES MD : BATCH
 ÜBERSETZUNG FEHLERFREI

ENDF PS&FTNCOMP (70.12) 1.48
 GNU.,BATCH

START PS&MONTSEGM (17.05)

ENDF PS&MONTSEGM (17.05) 9.22
 *STARTE,BATCH,DATEI=10-DS,DATEN=/ FREMDSTRING LÄNGER ALS 160 ZEICHEN

LISTE DER GENERATIONEN

GENERATION NR. 1

ZAHL DER EL. = 2
 ----- NAME LAUTET
 ----- X2
 ----- NAME LAUTET
 ----- Q2
 ----- KEINE ELEMENTBEZIEHUNG
 ----- Q1 M1

GENERATION NR. 2

ZAHL DER EL. = 3
 ----- NAME LAUTET
 ----- Q1
 ----- NAME LAUTET
 ----- X1
 ----- NAME LAUTET
 ----- X4
 ----- ELEMENTBEZIEHUNG ZWISCHEN
 ----- Q1 M2

 RUECKKOPPLUNG

GENERATION NR. 3

ZAHL DER EL. = 4
 ----- NAME LAUTET
 ----- X7
 ----- NAME LAUTET
 ----- X3
 ----- NAME LAUTET
 ----- Q2
 ----- NAME LAUTET
 ----- X2
 ----- KEINE ELEMENTBEZIEHUNG
 ----- Q1 M3

GENERATION NR. 4

ZAHL DER EL. = 5
 ----- NAME LAUTET
 ----- Q1
 ----- NAME LAUTET
 ----- X1
 ----- NAME LAUTET
 ----- X6
 ----- NAME LAUTET
 ----- X4
 ----- NAME LAUTET
 ----- Q2

----- ELEMENTBEZIEHUNG ZWISCHEN
 ----- Q1 M4

 RUECKKOPPLUNG

GENERATION NR. 5

ZAHL DER EL. = 6
 ----- NAME LAUTET
 ----- X1
 ----- NAME LAUTET
 ----- X7
 ----- NAME LAUTET
 ----- Q1
 ----- NAME LAUTET
 ----- X3
 ----- NAME LAUTET
 ----- Q2
 ----- NAME LAUTET
 ----- X2
 ----- ELEMENTBEZIEHUNG ZWISCHEN
 ----- Q1 M5

 RUECKKOPPLUNG

GENERATION NR. 6

ZAHL DER EL. = 7
 ----- NAME LAUTET
 ----- Q1
 ----- NAME LAUTET
 ----- X1
 ----- NAME LAUTET
 ----- X6
 ----- NAME LAUTET
 ----- X4
 ----- NAME LAUTET
 ----- X2
 ----- NAME LAUTET
 ----- Q2
 ----- NAME LAUTET
 ----- X3
 ----- ELEMENTBEZIEHUNG ZWISCHEN
 ----- Q1 M6

 RUECKKOPPLUNG

GENERATION NR. 7

ZAHL DER EL. = 8
 ----- NAME LAUTET
 ----- X6
 ----- NAME LAUTET

```

3 ----- X1
----- NAME LAUTET
----- X4
----- NAME LAUTET
----- X7
----- NAME LAUTET
----- Q1
----- NAME LAUTET
----- X3
----- NAME LAUTET
----- Q2
----- NAME LAUTET
----- X2
----- ELEMENTBEZIEHUNG ZWISCHEN
----- Q1 M7

----- X7
----- NAME LAUTET
----- Q1
----- NAME LAUTET
----- X3
----- ELEMENTBEZIEHUNG ZWISCHEN
----- Q1 M9

*****
RUECKKOPPLUNG
*****
GEAENDERT: &ST0DB.DS (0001.00)
STOP
ENDE BATCH 5.15

```

```

*****
RUECKKOPPLUNG
*****

```

GENERATION NR. 8

ZAHL DER EL. = 8

```

----- NAME LAUTET
----- X1
----- NAME LAUTET
----- X6
----- NAME LAUTET
----- X4
----- NAME LAUTET
----- X2
----- NAME LAUTET
----- Q2
----- NAME LAUTET
----- X7
----- NAME LAUTET
----- X3
----- NAME LAUTET
----- Q1
----- ELEMENTBEZIEHUNG ZWISCHEN
----- Q1 M8

```

```

*****
RUECKKOPPLUNG
*****

```

GENERATION NR. 9

ZAHL DER EL. = 8

```

----- NAME LAUTET
----- X2
----- NAME LAUTET
----- X6
----- NAME LAUTET
----- Q2
----- NAME LAUTET
----- X1
----- NAME LAUTET
----- X4
----- NAME LAUTET

```

```

DATUM: 28.01.74
AUF.-ZEIT: 18.38 UHR
END-ZEIT: 18.41 UHR
BEN: 4101101003 BRUE13
FKZ: WECK
KSB: 27 K
TSB: 50 K
PSB: 200 K
DRS: 100 S
RZS: 30 SEC
RK-ZEIT: 17.617 SEC
TRANSP.INDEX: 22.731 SEC

```

MINIMALWERTE DES ABSCHNITTES:

```

KSB: 27 K
TSB: 47 K
PSB: 160 K

```

LITERATURVERZEICHNIS

- /1/ R. Williams: A Survey of Data Structures for Computer Graphics Systems, ACM Computing Surveys, 3,1, March 1971, pp. 1-21
- /2/ J. Encarnaçao: Datenstrukturen für graphische Informationsverarbeitung, Fachtagung über Computer Graphics, Berlin, 19. bis 21. Oktober 1971, GI Bericht Nr. 2, Redaktion durch GMD, Bonn-Birlinghoven, pp. 15-49
- /3/ Ch. Köhler: Ein System zur Darstellung und Bearbeitung assoziativer Datenstrukturen, Diplomarbeit bei der Gesellschaft für Mathematik und Datenverarbeitung, Bonn-Birlinghoven, 1972
- /4/ D.E. Knuth: The Art of Computer Programming, Vol.3, 1973, Addison-Wesley Publ.Co.
- /5/ V.Y. Lum, P.S.T. Yuen, M. Dodd: Key-to Address Transform Techniques: Fundamental Performance Study on Large Existing Formatted Files, Communications of the ACM, Vol.14, April 1971, pp.228-239
- /6/ V. Glatzer: DATAS - Ein Systemkonzept zur Implementierung von assoziativen Datenstrukturen, Diplomarbeit am Institut für Informationsverarbeitung I, Technische Universität Berlin, 1971; Betreuer: W. Giloi und J. Encarnaçao
- /7/ W.R. Sutherland: The CORAL Language and Data Structure, Lincoln Lab. TR401, MIT, January 1966
- /8/ P.D. Rovner, J.A. Feldman: The LEAP Language and Data Structure, Proc. IFIP Cong. 1968, Vol.1, North-Holland Publ.Co., Amsterdam, pp.579-585
- /9/ M.F. Crick, R.A. Lorie, E.J. Mosher, A.J. Symonds: A Data-Base System for Interactive Applications, Report of the Cambridge Scientific Center, IBM, Cambridge (Mass.), No. G320-2058, July 1970

- /10/ M.F. Crick, A.J. Symonds: A Software Associative Memory for Complex Data Structures, Report of the Cambridge Scientific Center, IBM, Cambridge (Mass.), No. G320-2060, August 1970
- /11/ V. Glatzer, J. Encarnação: DATAS - Datenstrukturen in assoziativer Speicherung, Angewandte Informatik, Heft September 1972, pp. 417-424
- /12/ B. Gebhardt: Handbuch der Deutschen Geschichte, Band II, Stuttgart 1955 (8. Aufl.)

Die folgenden Firmenschriften wurden beim Schreiben der Programme benutzt; sie wurden im vorliegenden Bericht jedoch nicht eigens zitiert:

- /13/ Control Data: 3100/3200/3300/3500 Computer Systems USASI FORTRAN MASTER/MSOS Reference Manual, Pub. No. 60281400, St. Paul (Minn.), 1970
- /14/ Control Data: 3170/3300/3500 Computer Systems MASTER Version 3.3 Library Generation and Maintenance Reference Manual, Pub.No. 60373100, St. Paul (Minn.), 1972
- /15/ Control Data: 3300/3500 MASTER Diagnostic Handbook, Pub.No. 60206800A, St. Paul (Minn.), 1968
- /16/ Varian Data Machines: Varian V73 System Handbook, Doc. No. 98A 9906 010, Irvine (Cal.), June 1972
- /17/ Varian Data Machines: Varian Software Handbook, Doc.No. DP-120-12/71-5, Irvine (Cal.), 1971
- /18/ Varian Data Machines: VORTEX Reference Manual, Doc.No. 98A9952 101, Irvine (Cal.), September 1972
- /19/ Telefunken Computer: System TR440, FORTRAN Sprachbeschreibung, Best.Nr. N31.D1.03, Ausgabe 0172-VS1/Ni, Konstanz, 1971
- /20/ Telefunken Computer: TR440 Programmbeschreibung, FORTRAN Hilfsprogramme, Beschr.-Nr. N31.E1.11, Ausgabe 0871-P33/Käab, Konstanz, 1971

- /21/ Telefunken Computer: System TR440, Programmiersystem
Einführung, Best.-Nr. 440.BO.02, Ausgabe 0173-VS1,
Konstanz, 1973
- /22/ Telefunken Computer: System TR440, Kommandosprache,
Best.-Nr. N31.DO.01, Ausgabe 0272 ES., Konstanz, 1972
- /23/ Telefunken Computer: Programmbibliothek, Mitteilungen für
Benutzer zur MV14, Konstanz, Oktober 1973
- /24/ digital equipment: pdp 11, DOS 11 Platten-Betriebssystem,
Druckschrift G11B 1172 020, München, 1973