INSTITUT FÜR
INFORMATIK

Mitteilung 34

Erweiterung des DECSystem-10 PASCAL-Compilers um eine Möglichkeit zur Erzeugung eines Post-Mortem-Dump

Bernhard Nebel, Bernd Pretschner

IfI-HH-M34/76

D 2000 HAMBURG 13
SCHLÜTERSTRASSE 70

Inhaltsverzeichnis
==================

0. Vorgeschichte

Der Anstoss zu dieser Arbeit war unsere "fixe Idee", uns etwas mit dem PASCAL-Compiler (auf dem DECSystem-10) zu befassen.
Als wir das erste Mal bei Herrn Nasel vorsprachen, hatten wir den Wunsch, irgendetwas zu machen. Erst im Laufe mehrerer Gespraeche (mit und ohne Herrn Nasel) entwickelten sich die in den Abschnitten 1 und 2 beschriebenen Zielvorstellungen [0]. Das Reizvolle an diesem Projekt war die Erfahrung der Einarbeitung in ein uns unbekanntes groesseres System und dessen Modifikation.
Die Abschnitte 1 bis 4 sind das Ergebnis einer Analyse des Problemraumes und eine Beschreibung der projektierten Aenderungen. Diese Abschnitte sind aufgrund einer Forderung von Herrn Nasel schriftlich formuliert vorgelegt worden, bevor die Implementation begonnen wurde.
In den Abschnitten 5 und 6 haben wir die Ergebnisse und Erfahrungen der Implementation festgehalten.

Im nachhinein koennen wir sagen, dass wir diese Methode der vorausgehenden schriftlichen Analyse eines Problemkomplexes und die die Implementation begleitenden und abschliessenden Dokumentationen als vorteilhaft erlebt haben. Auch kann man sich anhand einer solchen Dokumentation erheblich leichter in Programmsysteme einarbeiten [1, S. 385ff].
Unser Lernerfolg bei diesem Projekt besteht nicht nur aus einer Uebersicht ueber das PASCAL-Compiler-Programmsystem, sondern zu einem grossen Teil in der positiven Erfahrung mit der oben dargestellten Arbeitsmethode, mit welcher wir auch in zukuenftigen groesseren Projekten zu arbeiten beabsichtigen.

0.1 Zeitplan

15. Januar - 15. Februar 1976
: Einarbeitung in den Problemraum und Erstellung der Analyse (Abschnitte 1 bis 4)

15. Februar - 15. Maerz 1976
: Urlaub

15. Maerz - 4. April 1976
: Implementation der in der Analyse beschriebenen Aenderungen, sowie Planung und Implementation mehrerer nicht im Detail in den Abschnitten 3 und 4 beschriebener Erweiterungen (Abschnitt 5)

4. April - 13. April 1976
: Pruefen des neuen Systemes und Suche nach "pathologischen" Fehlern

bis Mitte Juni 1976
: Abschliessende Dokumentation (Abschnitte 6 und 0)

1. Vorbemerkungen

1.1 Zielvorstellungen

Aehnlich wie in der Zuericher CDC-PASCAL-Implementation [2] verwirklicht, wollen wir einen Post-Mortem-Dump (im folgenden als PASPMD bezeichnet) fuer PASCAL-Programme realisieren. Dieser Dump soll enthalten:

a) Fehlerdiagnose und Angabe der Zeilennummer des Quellprogrammes, bei der die Programmausfuehrung abgebrochen wurde.
b) Prozedur-Aktivierungs-Rueckverfolgung (Backtracing)
c) Ausgabe der Variablen mit Bezeichner und Wert in PASCAL- Notation.

1.2 Begruendung

Zur Zeit existiert schon ein Pruefhilfen-System fuer PASCAL-Programme (PASDDT) (siehe [3],[5]) , welches aber nur interaktiv arbeitet. Aber gerade Anfaenger, die im Stapel-Betrieb arbeiten muessen, machen oft Programmierfehler, welche zum Abbruch des Programmlaufs fuehren koennen (eigene Erfahrungen). Sie erhalten dann u.U. einen Dump auf Maschinenebene, der einem Anfaenger kaum verwertbare Hinweise gibt, wo der "Fumel" denn nun eigentlich steckt.
Da Anfang des SS 76 wieder eine Einfuehrungsvorlesung beginnt, und Herr Brauer uns bestaetigt hat, dass die Programmiersprache PASCAL gelehrt wird, folgt:

a) unser Projekt ist nicht sinnlos!
b) wir sollten unseren Zeitplan so gestalten, dass die wichtigsten PASPMD-Massnahmen Anfang des SS 76 implementiert sind.

1.3 Ausgangspunkt

Wie oben erwaehnt, existiert schon ein interaktives Pruefhilfen-System (in der Fachliteratur auch Debugsystem genannt) fuer PASCAL - Programme. Dieses Debugsystem wollen wir so modifizieren, dass es zusaetzlich zur interaktiven Benutzung auch den unter 1.1 aufgestellten Zielvorstellungen entspricht.

2. Spezifizierung der Zielvorstellungen

2.1 Welche Laufzeitfehler sollen den PASPMD aktivieren?

2.1.1 Welche Fehler werden bisher von der Laufzeitunterstuetzung diagnostiziert?

Fehlermeldung	fuehrt zum Uebergang ins PASDDT:
a) stack overruns heap	nein
b) error in fortran procedure	nein
c) heap overruns stack	nein
d) attempt to read beyond eof	nein
e) output error: disk space exhausted	nein
f) rewrite required	nein
g) no access to or no disk space for..	nein
h) input data error in file	nein
i) array index out of bounds	Ja
k) scalar out of range	Ja

Laengere Diskussionen ergaben, dass wir die Fehlerbehandlung folgendermassen modifizieren wollen:

Fehler a, c, d, e, h, (i, k) sollten zu einem Aufruf des PASDDT-System fuehren, und zwar entweder in den Pruefzustand bei interaktivem Betrieb oder in den PASPMD bei Stapelverarbeitung, sofern bei der Compilation durch Angabe der Debug-Option (D+) das PASDDT, bzw. das PASPMD System angefordert worden ist.
Dieses gilt fuer Fehler a nur eingeschraenkt:
es sollte abgeprueft werden, ob der Fehler in der Initialisierungsphase des Programmes auftrat, weil der Benutzer beim Start des Programmes zu wenig Arbeitsspeicher angefordert hat (siehe [7], SSAVE-, SAVE-, RUN - command).

2.1.2 Laufzeit-Fehlermeldungen des Monitors

Die haeufigsten Fehlermeldungen des Monitors waehrend eines Programmlaufes sind:

i) ill mem ref, d.h. Zugriff auf nicht definierte Speicherbereiche,

und im Stapelverarbeitungs-Betrieb noch

ii) Time limit exceeded

Fehlerursachen sind meist Programmierfehler. Bei i) handelt es sich haeufig um Fehler im Umgang mit Pointern. Bei ii) sind es meistens Tot-Schleifen.

Wir halten es fuer sinnvoll, die beiden oben erwaehnten Fehler im Stapel-Betrieb bei gesetzter Debug-Option abzufangen.

2.2 Was soll ausgegeben werden?

Folgende Probleme stellen sich hier :

a) Welche Daten sollen ausgegeben werden, um den Benutzer einerseits nicht mit einem Berg von Informationen zu erschlagen, andererseits aber genug Informationen zu geben, damit er den Fehler finden kann?

b) Wie kann man die Daten uebersichtlich und platzsparend auflisten ?

Das unter a) gesagte ist ein generelles Problem, auf das z.B. in ([1],Debugging and Testing,P.C. Poole), [3] und [8] eingegangen wird, waehrend b) ein rein ausgabetechnisches Problem ist und hier nicht in allen Einzelheiten ausgefuehrt werden muss.

2.2.1 Fehlerdiagnose und Angabe der Zeilennummer, bei der die Programmausfuehrung abgebrochen wurde

2.2.2 Prozedur-Rueckverfolgung
(Trace-Funktion in PASDDT)

Hierbei ist darauf zu achten, dass der Ausgabe-Algorithmus abbricht, wenn innerhalb des Laufzeitkellers die Aktivierungsrecordverknuepfungen vom Benutzer zerstoert worden sind, was z.B. auftreten koennte, wenn die Laufzeittest-Option zur Ueberpruefung von Ueberschreitungen der Arraygrenzen abgeschaltet ist. Das Abbrechen des Rueckverfolgungs-Algorithmus, der anhand der Aktivierungsrecordverknuepfungen die Quellprogrammzeilennummer der Prozedur-Aktivierung ausgibt, kann garantiert werden, wenn jeweils gepruefft wird, ob der Verweis auf die Ruecksprungadresse der rufenden Prozedur echt kleiner ist als die Adresse, in der der Verweis eingetragen ist.
(Zum Aufbau des Laufzeitkellers siehe [9])

2.2.3 Ausgabe der deklarierten Variablen

Alternative i) Ausgabe der z.Z. gueltigen Variablen

Alternative ii) Ausgabe aller Variablen der z.Z. aktivierten Funktionen und Prozeduren.

Fuer die Alternative i) sprechen:

geringerer Aufwand (Papier!),
Konsistenz mit der PASCAL Konvention zu den Gueltigkeitsbereichen von Bezeichnern

Fuer die Alternative ii) sprechen:

die Ursache fuer den Fehler koennte in einer Prozedur liegen, die auf einem dynamisch hoeheren Niveau liegt, deren Variablen aber zur Zeit nicht gueltig sind.

Als Vorschlag wuerden wir einen Kompromiss aus beiden Alternativen machen:

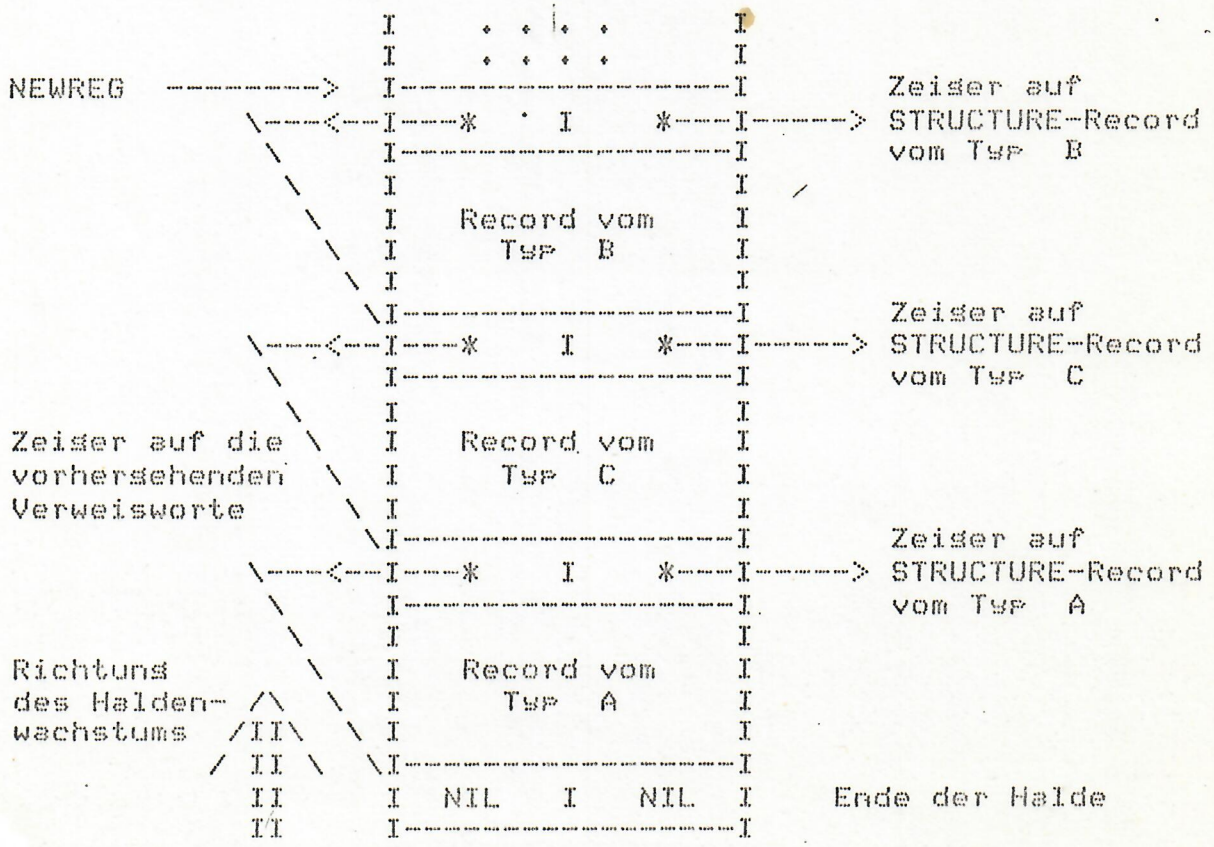
Zuerst eine Ausgabe der Variablen der dynamischen Prozedurverschachtelungen bis zu einer maximalen Tiefe von 10 Prozeduren;
danach, falls die dynamische Schachtelungstiefe groesser als 10 ist, eine Ausgabe der Variablen der statischen Prozedurverschachtelung.
(Beide Angaben beginnen jeweils bei der unterbrochen Prozedur)

2.2.3.1 Formatierung
Zur Zeit wird von PASDDT eine skalare Variable, bzw. eine skalare Komponente einer strukturierten Variablen, pro Zeile ausgegeben. Da uns beim PASPMD 132 Zeichen pro Zeile zur Verfuegung stehen (Ausgabe auf dem Zeilendrucker!), muessen wir uns etwas einfallen lassen, um nicht als Papierverschwender "angeprangert" zu werden.

2.2.3.2 Ausgabe von strukturierten Variablen
a) Was soll bei Records ohne Tagfieldidentifier ausgegeben werden?
b) Ist es sinnvoll, bei groesseren Arrays alle Komponenten auszugeben?
Hier gilt dasselbe wie bei 2.2.3.1

2.2.4 Ausgabe der dynamisch erzeugten Variablen (Haldeninhalt)
Dieser Teil des PASPMD ist wahrscheinlich am schwierigsten zu implementieren, aber u.U. nicht der wichtigste. Trotzdem wollen wir auch hier einen Vorschlag machen:
Bei der Prozedur NEW wird zu jeder dynamisch erzeugten Variablen zusaetzlich ein Wort reserviert, das
erstens einen Verweis auf das Verweiswort des zuvor erzeugten Records enthaelt und
zweitens einen Verweis auf den STRUCTURE-Record des Typs der neu alloziierten Variable.

Beispiel:



Bei der Aussabe der Halde muss dafuer sesorst werden, dass keine Totschleifen oder Ill-mem-ref's auftreten, die durch fehlende Verweisworte oder deren Ueberschreibuns verursacht werden koennten.

Es ist allerdinss noch ein Problem, die Verweise auf die STRUCTURE-Records zu senerieren, da die Bezeichner-Baeume erst am Ende einer Prozedur seschrieben werden. Bei der Codeerzeusuns sind die Adressen der STRUCTURE-Records folslich noch nicht bekannt. Diese Adressen koennten mit einem Verfahren analos dem bei Vorwaertsspruensen verwendeten nach erfolster Uebersetzuns vor Aussabe des Prozedurrumpfes einsesetzt werden.

3. Analyse des PASCAL-DEBUG-Systems (PASDDT)

Das PASDDT musste von den Autoren durchgearbeitet werden, um einen Ueberblick ueber die bei der Implementation von PASPMD zu bewaeltigenden Aufgaben zu bekommen. Dabei ist die Dokumentation dieses Durcharbeitens soweit wie moeglich unabhaengig von der Dokumentation des interaktiven PASDDT-Systems [3] angefertigt worden.
An dieser Stelle moechten wir Peter Putfarken danken, dass er uns in die "Tricks und Schliche" seines Programmes einfuehrte und uns bei Verstaendnisschwierigkeiten mit Rat und Tat zur Seite stand.

Das PASDDT-System besteht aus folgenden Komponenten:
a) Der PASCAL-Compiler, der relozierbaren Code erzeugt ("PASREL"-Compiler, siehe Abschnitt 3.1)
b) Die zum PASCAL-Compiler gehoerige Laufzeitunterstuetzung (siehe Abschnitt 3.2)
c) Die PASDDT-Laufzeitunterstuetzung (siehe Abschnitt 3.3)
d) Das eigentliche PASDDT-Programm (siehe Abschnitt 3.4)

Die folgenden Abschnitte, in denen die Funktionen der einzelnen Komponenten erklaert werden, setzen voraus, dass der Leser mit der grundlegenden Struktur des DECSystem-10 PASCAL-Compilers vertraut ist.

3.1 Der PASCAL-Compiler

3.1.1 Realisierung von zeilenorientierten Stops

Der Compiler schreibt in den Instruktionscode vor den Code jeder Anweisung einer neuen Zeile (des Quellprogramm-Datensatzes) eine NOOP-Instruktion, die als LINERecord bezeichnet wird, welcher die Differenz der Zeilennummern zwischen aktueller und letzter bearbeiteter Zeile (u.U. auch den absoluten Wert) und einen Pointer auf den letzten LINER enthaelt (siehe PUTLINER). Auf aehnliche Weise werden PAGERecords in den Instruktionscode eingebaut. Im Pruefzustand koennen anstelle der LINER-NOOPs Aufrufe des PASDDT-Systems eingetragen werden (siehe 3.1.2). Die kurzzeitig notwendige Aenderung des Protection-Bits des High-Segments wird durch die vom Compiler bereitgestellte Prozedur PROTECTION ermoeglicht.

3.1.2 Einbau von Verweisen zum PASDDT

Die Initialisierungsprozedur des PASDDT wird waehrend der Initialisierungsphase des Programms angesprungen (siehe LEAVEBODY: PUSHJ TOPP,INDEB.).
Der Aufruf von PASDDT waehrend des Programmlaufes (STOP) wird durch Ersetzen eines LINER erzwungen. Und zwar durch den Inhalt der Zelle 141B (oktal), in welche der Compiler ein PUSHJ TOPP,EXDEB. eingetragen hat (siehe MCCODE / MCLIBARY).

3.1.3 TTYinput / TTYoutput

Da das PASDDT-Programm interaktiv arbeitet, werden auf Jeden Fall die Teletypefiles eroeffnet (siehe LEAVEBODY).

3.1.4 Speicherung der Bezeichner mit dazugehoerigen Strukturen

Der Bezeichnerbaum jeder Prozedur wird am Ende derselben in das High-Segment des Benutzerprogramms kopiert (siehe Prozeduren COPYSTP, COPYCTP, COPYCSP). Vor dem Prozedurcode wird ein Verweis auf die Wurzel dieses Baumes eingetragen. Der Bezeichner des Wurzelrecords besteht aus 10 Leerzeichen und der NEXT-Pointer desselben zeigt auf ein IDENTIFIER-Record mit dem Prozedurnamen.

3.1.5 Weitere Parameteruebergaben

Im Record DEBUGENTRY werden Zeiger auf den GLOBALIDTREE, STANDARDIDTREE, CHAR-, REAL- und INTEGER-Records uebergeben. Die Adresse des DEBUGENTRY-Records wird in Zelle 142B (oktal) abgelegt. (siehe Prozedur MCCODE)

3.2 Laufzeit-Unterstuetzung fuer PASCAL-Programme

Bei den Fehlern "scalar out of range" und "array index out of bounds" erfolgt ein Aufruf von PASDDT, wenn bei der Compilation die Debug-Option gesetzt worden ist. Dies kann zur Laufzeit festgestellt werden, indem man die Zelle .JBDDT im JOBDATA-Bereich (siehe [6], monitor calls, section 1.2.1) auf 0 testet. Dann wird der Inhalt der Adresse .JBDDT dekrementiert und das Resultat als Anfrungsadresse, die dann dem Label ERRDB. in der PASDDT-Laufzeitunterstuetzung entspricht (siehe 3.3), benutzt.

3.3 PASDDT-Laufzeit-Unterstuetzung

Die PASDDT-Laufzeitunterstuetzung stellt ein Bindeglied zwischen Benutzer- und PASDDT-Programm dar. Sie enthaelt 4 Eintrittspunkte, von denen 2 explizit angegeben werden muessen (fuer das Systemprogramm LOADER). Ausser den oben erwaehnten Eintrittspunkte INDEB. und EXDEB. (siehe 3.1.2) existieren DDTDB. (Anfrungspunkt bei DDT-Kommando) und ERRDB. (siehe 3.2).

Beim ersten Aufruf (INDEB.) des PASDDT wird ein eigener PASDDT-Speicherbereich reserviert. Allen Eintrittspunkten gemeinsam ist:

Retten der Register,
Setzen der STATUS-Variablen,
Feststellen der Rueckkehr-Adresse,
Aufruf des PASDDT-Programms,
Restaurieren der Register und
Rueckfrung ins Benutzerprogramm bzw. Frung zum Ende.

3.4 Das PASDDT-Programm

Das eigentliche PASDDT-Programm ist eine externe PASCAL-Prozedur. Dabei ergibt sich folgendes Problem: PASCAL ist keine System-Implementationssprache, aber innerhalb des PASDDT-Programmes muss z.B. der Laufzeitkeller und die Halde inspiziert, interpretiert und modifiziert werden. Um die von Benutzerprogramm zu Benutzerprogramm unterschiedlichen Datenstrukturen verarbeiten zu koennen, muss das PASDDT-Programm die maschineninterne Darstellung interpretieren. Dazu sind Datenstrukturen notwendig, welche sehr von der Rechenanlage abhaengen, auf der das Programm laeuft. Trotzdem sind die Autoren der Meinung, das PASCAL auch fuer solche Probleme angemessen ist, da das Programm strukturierter ist, als wenn es in einer Assemblersprache geschrieben worden waere. Haette man das Problem mit Hilfe des MACRO10-Assemblers programmiert, waere wahrscheinlich ein erheblich groesserer Zeitaufwand noetig gewesen, das Programm zu verstehen und zu aendern.

3.4.1 Aufgabe des Programmes

Es werden Eingaben des Benutzers eingelesen, auf syntaktische Korrektheit gepruefet und verarbeitet.

3.4.2 Datenstrukturen

Die Strukturen IDENTIFIER, STRUCTURE und ATTR sowie die dazugehoerigen Typdeklarationen sind dem Compiler entnommen. Dieses ist notwendig, weil die Bezeichner mit den dazugehoerigen Beschreibungen vom Compiler uebernommen werden (siehe 3.1.4).

Im folgenden beschreiben wir die in unseren Augen wesentlichen Datenstrukturen, die zum Verstaendnis der Arbeitsweise des Debugsystems notwendig sind.

```
ACR = ^AKTIVIERUNGSRECORD;
AKTIVIERUNGSRECORD = ARRAY[0..0] OF INTEGER;
```

ACR wird benutzt, um indirekte und indizierte Adressierung von Maschinenworten in PASCAL zu realisieren. Das Ueberschreiten der Array-Grenzen (BASIS^[0-1] oder BASIS^[0+20B]) wird vom Compiler nicht bemerkt, da der Index ein Ausdruck ist (BASIS^[20B] wuerde waehrend der Kompilation beanstandet werden). Die Laufzeitpruefung auf Ueberschreitung der Index-Grenzen muss deshalb abgeschaltet sein.

```
TYPE DYNENTRY = RECORD
                  REGISTERS   :ACR;            (* 140B *)
                  STOPPY      :INTEGER;        (* 141B *)
                  ENTRYPTR    :^DEBUGENTRY;    (* 142B *)
                  STACKBOTTOM:ACR;             (* 143B *)
                  STATUS      :DEBUGSTATUS ;   (* 144B *)
                END;

VAR ENTRY2 :  DYNENTRY;
```

In der Prozedur INIT wird der Inhalt der Adressen 140B bis 144B in ENTRY2 kopiert. Diese Speicherzellen, die im Anschluss an den DECSystem-10 spezifischen JOBDATA-Bereich von 0 bis 137B liegen, dienen zur Uebergabe von Parametern des Benutzerprogrammes an das PASDDT-Programm.
Bedeutung der Record-Felder:

REGISTERS: Verweis auf den Anfang des PASDDT-Speicher-Bereiches, in dessen ersten 16 Worten die geretteten Registerinhalte des Benutzerprogramms stehen (siehe 3.3).

STOPPY: hier steht der Maschinenbefehl PUSHJ TOPP, EXDEB. (siehe 3.1.2)

ENTRYPTR: Pointer auf das Record des Typs DEBUGENTRY (siehe 3.1.5).

STACKBOTTOM: Pointer auf den Anfang des Laufzeit-Kellers des Benutzerprogrammes; gesetzt in der Initialisierungsphase des Programmes; wird in TESTGLOBALBASIS verwendet.

STATUS: Die PASDDT-Laufzeitunterstuetzung traegt in STATUS die Rueckkehradresse in das Benutzerprogramm und die Art (Initialisierung, Stop, Laufzeitfehler, DDT-Kommando) des Aufrufes des PASDDT-Systems ein (siehe 3.3).

Die anderen Datenkonstrukte sind im wesentlichen leicht verstaendlich, da selbstdokumentierend. Nur GATTR sei noch kurz erlaeutert, da diese Variable im folgenden haeufig erwaehnt wird. Im PASDDT-Programm wird mit GATTR der aktuell zu bearbeitende Ausdruck beschrieben. GATTR ist vom Typ ATTR, der im PASDDT-Programm folgendermassen definiert ist:

```
ATTRKIND = (CST,VARBL,EXPR);
ATTR = RECORD
         TYPTR: STP;
         CASE KIND: ATTRKIND OF
           CST,
           EXPR:  (CVAL: VALU);
           VARBL:(PACKFG: BOOLEAN;
                  GADDR: ADDRRANGE;
                  GBITCOUNT: BITRANGE)
       END;
```

Bedeutung der einzelnen Record-Felder:

TYPTR: Verweis auf den STRUCTURE-Record des aktuellen Ausdrucks

KIND: Art des aktuellen Ausdrucks (Konstante, Ausdruck, Variable)

im Falle KIND=CST,EXPR :

CVAL: Wert des Ausdrucks (bei Strins-Konstanten ist CVAL der Wert des Pointers auf den Strins)

Im Falle KIND=VARBL :

PACKFG: Gibt an, ob die Variable gepackt ist

GADDR: Absolute Adresse der Variablen

GBITCOUNT: Wenn es sich um eine gepackte Variable handelt, so wird auf GBITCOUNT gezaehlt, wieviele Bits eines Maschinenwortes schon abgearbeitet wurden.

3.4.3 Programmstrukturen

Im folgenden beschreiben wir die nicht kommentierten und nicht ohne weiteres verstaendlichen Prozeduren des PASDDT.

3.4.3.1 PROCEDURE INSYMBOL

Liest naechstes Basissymbol ein. Seiteneffekte auf folgende Variablen:

SY:	Art des Grundsymbols
CH:	naechstes Zeichen
CHCNT:	Anzahl der einselesenen Zeichen der aktuellen Zeile
ID:	Bezeichner
VAL:	Wert bei Konstanten (CHAR, REAL, INTEGER)
LGTH:	Laenge der letzten Zeichenkette (strins)
STRING:	Verweis auf einselesene Zeichenkette
STRINGPTR:	Verweis auf STRUCTURE-Record der Zeichenkette
STRINGINDEX:	Beschreibung des Index, gehoert zu STRINGPTR

3.4.3.2 FUNCTION IDTREE

Die Aufgabe der Funktion ist, zu einer Prozedur, auf deren Aktivierungsrecord die globale Variable BASIS zeigt, den Verweis auf ihren Bezeichnerbaum zu finden. Die Wirkung ist zwar im Programm beschrieben, aber das Verfahren nicht. Es wird aus dem Keller des Benutzerprogramms die Ruecksprungadresse der unterbrochenen Prozedur geholt. In der rechten Haelfte der vorherigen Instruktion, welche den Aufruf (PUSHJ TOPP,...) der Prozedur bewirkte, steht also die Anfangsadresse des Codes der gerufenen (unterbrochenen) Prozedur. Vor dem eigentlichen Prozedurcode stehen bis zu 6 Instruktionen der Form

```
HRR BASIS, -1(BASIS)
```

um den "static link" in Abhaengigkeit von der Deklarationsebene der rufenden Prozedur richtig zu setzen. Davor steht der schon unter 3.2.1.4 erwaehnte Verweis auf den Bezeichner-Baum der Prozedur. Also muss man u.U. bis zu 6 Instruktionen vor die Anfangsadresse gehen, um den Verweis zu finden. Er unterscheidet sich von den HRR-Befehlen dadurch, dass das erste Bit (Vorzeichen-Bit) Null ist (HRR hat den OP-Code 101100 0 und wird als negative INTEGER-Zahl interpretiert, waehrend das linke Halbwort des Verweisswortes 0 ist).

Kommentar:

Auf Grund eigener Schwierigkeiten beim Verstehen haben wir diese Funktion so ausfuehrlich erklaert.

3.4.3.3 PROCEDURE SEARCHSECTION

Sucht den Bezeichnerbaum einer Prozedur nach einem Bezeichner mit dem in ID abgelegten Namen durch.

3.4.3.4 PROCEDURE SEARCHID
Sucht mit Hilfe von SEARCHSECTION alle sueltigen Bezeichner-Baeume nach dem in ID abgelegten Bezeichner ab.

3.4.3.5 FUNCTION NEXTBYTE
Uebergibt den Wert des auf GATTR beschriebenen Bytes der Laenge FBITSIZE (Seiteneffekt auf GATTR.GBITCOUNT, u.U. GATTR.GADDR).

3.4.3.6 FUNCTION PUTNEXTBYTE
Schreibt den Wert FVAL in das naechste Byte der Laenge FBITSIZE, wie es von GATTR spezifiziert wird (Seiteneffekte wie 3.4.3.5).

3.4.3.7 PROCEDURE GETFIELD
Veraendert GATTR so, dass mit NEXTBYTE oder PUTNEXTBYTE auf das Recordfeld (d.h. auf dessen Wert) zugegriffen werden kann. Der Parameter ist ein Verweis auf einen IDENTIFIER-Record des anzusprechenden Record-Feld-Bezeichners.

3.4.3.8 PROCEDURE SELECTOR
Selektiert die angegebene Komponente (wird in SELECTOR eingelesen) einer strukturierten Variablen, indem GATTR so bestimmt wird, dass beim naechsten Aufruf von NEXTBYTE bzw. PUTNEXTBYTE die Komponente erreicht werden kann.

3.4.5.9 PROCEDURE VARIABLE
Bestimmt in Abhaengigkeit des eingelesenen Bezeichners GATTR.

3.4.3.10 PROCEDURE STOPMESSAGE
S o l l die zum Programmzaehler gehoerige Zeilennummer und Seitennummer ausgeben; gibt sinnlose Zeilennummern bei Unterbrechungen und Uebergang ins PASDDT, wenn das Programm sich in der Laufzeitunterstuetzung oder sich bereits im PASDDT-Zustand befindet.

3.4.3.11 PROCEDURE TRACEOUT
Hangelt sich durch die dynamischen Prozedurverschachtelungen (dynamic link) und gibt dabei eine Prozeduren-Rueckverfolgung mit den Zeilen- und Seitennummern des Aufrufortes jeder aktivierten Prozedur aus.

3.4.3.12 PASDDT-Prozedurkoerper
Eigentliche Verarbeitung (interaktiv) der vom Benutzer eingegebenen Befehle.

Nachsatz:
Wir meinen, dass der geneigte Leser die oben nicht erwaehnten Prozeduren wegen der guten Strukturierung des Programms und der geeigneten Bezeichnerwahl im Anschluss an die hier skizzierten Erlaeuterungen ohne groessere Schwierigkeiten ueberschauen kann.

4. Projektierte Aenderungen

4.1 Im PASCAL-Compiler

4.1.1 In LEAVEBODY

Der Code fuer das Hauptprogramm muss so geaendert werden, dass auf jeden Fall der TTYOUTPUT-File eroeffnet wird (fuer die Ausgaben des PASDDT-Programms in den Log-File) und der TTY(-Input) File nicht initialisiert wird, wenn das Programm im Stapelverarbeitungsbetrieb laeuft und nicht vom Pseudo-TTY (PTY) liest.

4.1.2 In ENTERBODY

Um auch bei dem Fehler "Kellerueberlauf" (stack overruns heap) einen Uebergang ins PASDDT mit Rettung des PC's in AC0 zu erreichen, muss mit JSP statt JRST in die PASCAL-Laufzeitunterstuetzung gesprungen werden. Dafuer muss an der entsprechenden Stelle die Codeerzeugung geaendert werden.

Generelle Bedenken bei Abfragen dieses Fehlers:

Bei diesem Laufzeit-Fehlerhalt steht der PC vor dem ersten LINER des Prozedurkoerpers und nach dem letzten LINER des vorherigen Prozedurkoerpers, man wuerde also eine unsinnige Zeilennummer als Stopnummer erhalten. Um dieses zu verhindern, koennten wir einen LINER vor dem Sprung nach CORERR setzen. Dabei laufen wir aber Gefahr, dass Benutzer bei diesem LINER einen STOP setzen und die formalen Parameter abfragen, die an dieser Stelle noch nicht die Werte der aktuellen Parameter tragen.

Eine andere Moeglichkeit bestaende darin, die Aktivierungsadresse als Stopadresse zu nehmen und BASIS zurueckzusetzen.

4.2 Aenderungen in der Laufzeitunterstuetzung fuer PASCAL-Programme

Prinzipiell soll bei Laufzeit-Fehlern (s. Zielvorstellungen) ein Sprung ins PASDDT- bzw. PASPMD-System (bei gesetzter Debug-Option) erfolgen. Deshalb fuehren wir in der PASCAL-Laufzeitunterstuetzung eine neue Marke PASDDT ein, die vor jenen Anweisungen steht, in denen geprueft wird, ob PASDDT mit geladen worden ist (siehe auch 3.2), um es gegebenenfalls anzuspringen (8. Zeile von WRTPC).

4.2.1 NEWERR

Um den Konventionen der PASDDT-Laufzeit-Unterstuetzung zu entsprechen, muss die Stopadresse (in diesem Fall die Rueckkehradresse) in AC0 geladen werden: MOVE AC0,(TOPP)

4.2.2 RETRY

(fuer NEWERR und CORERR) Ansprung von PASDDT (siehe 4.2.1) durch JRST PASDDT.

4.2.3 GETEOF
Bei dieser Fehlermeldung koennen wir im Augenblick nicht feststellen, in welcher Zeile des Hauptprogramms wir stehen. Deshalb wird AC0 auf -1 gesetzt

```
SETO AC0,0
```

Ausserdem JRST END durch

```
JRST PASDDT
```

ersetzen.

4.2.4 PUTERR
Hier gilt das gleiche wie fuer GETEOF.

4.3 In der PASDDT-Laufzeit-Unterstuetzung
4.3.1 Zusaetzliche Konstanten

```
DUMPTIME    = 144;  time for dump in jiffies
.GTLIM      = 40 ;  argument for GETTAB
.JBSA       = 120;  adresses in
.JBAPR      = 125;  the JOBDATA-
.JBCNI      = 126;  area
.JBTPC      = 127
STACBO      = 143;  LH:  it is a timesharing job
                ;  RH:  stackbottom
```

4.3.2 ERRDB1
Um zu verhindern, dass beim Fehler "Stack overruns Heap" zu Programmbeginn (vgl. 2.1.1) auch in den Pruefzustand uebergegangen wird, muss dieser Fall speziell identifiziert und abgefangen werden.

Durch den JSP-Sprung zu CORERR wird der PC in AC0 eingetragen. Ist die Differenz zwischen Programmstartadresse und diesem PC gleich 6, so handelt es sich um den Fehler "Stack overruns Heap" bei Programminitialisierung.

```
ERRDB1:SUBI     AC0     ,6
       HLL      AC0     ,.JBSA
       CAMN     AC0     ,.JBSA
       JRST     END
       ADDI     AC0     ,6
       HRLI     AC0     ,3
                ....    .....
```

4.3.3 Abfangen von "Ill Mem Ref" und "Time Limit Exceeded"
Damit bei Zugriffen auf nicht definierte Speicherbereiche ("Ill Mem Ref") und bei Laufzeitueberschreitungen ("Time Limit Exceeded") in das PASDDT- bzw. PASPMD-System uebergegangen wird, rufen wir das APRENB-UUO auf (Siehe [6],monitor calls,section 3.1.3.1). Das erfordert den Eintrag einer Adresse in die Speicherzelle .JBAPR, zu der gesprungen wird, falls ein Fehler der Art auftritt, die beim Aufruf des APRENB-UUO angegeben wurde (siehe 4.3.4). Eintrag der benoetigten Adresse:

```
LOC     .JBAPR
XWD     0       ,APRINT
RELOC   400000
```

4.3.4 Initialisierung des APRENB-UUO

Vor dem ersten Sprung ins PASDDT (in INDEB.) wird INIAPR (PUSHJ TOPP,INIAPR) aufgerufen, in der folgendes gemacht wird:

Es wird gepruefft, ob das zu exekutierende Programm in der Stapelverarbeitung abgewickelt wird. Wenn nein, wird eine 1 (TRUE) in die linke Haelfte von Adresse 143 (rechte Haelfte enthaelt STACKBOTTOM, siehe 4.4.1.1) geschrieben, welche im PASDDT-Programm abgefragt wird. Bei Nicht-Stapel-Verarbeitung wird das APRENB-UUO nur fuer ill-mem-ref-Fehler aufgerufen. Bei Stapelverarbeitung wird eine interne Zeitgrenze (Time-limit) berechnet (System-time-limit minus DUMPTIME) und in LIMIT abgespeichert. Ausserdem wird das APRENB-UUO fuer Clock-flag und ill-mem-ref aufgerufen.

4.3.4.1 Code zu 4.3.4

```
INIAPR:MOVE    AC1     ,EXWD -1,,GETLIM
       GETTAB AC1     ,
       HALT                            ;  error return
       TLNN   AC1     ,400             ;  test if batch-Job
       JRST   NOTBAT                   ;  no
       TLZ    AC1     ,777740          ;  bit 0-12 to zero
       SUBI   AC1     ,DUMPTIME        ;  internal time-limit
       IMULI  AC1     ,24              ;  Jiffies to msec
       MOVEM  AC1     ,LIMIT           ;  stores time-limit
       MOVEI  AC1     ,21000           ;  argument for APRENB
                                       ;  ill-mem-ref and .
                                       ;  clock flag enable
       APRENB AC1     ,
       POPJ   TOPP    ,
NOTBAT:MOVEI  AC1     ,1               ;  this Job is
       HRLM   AC1     ,STACKBO         ;  a timesharing-Job
       MOVEI  AC1     ,20000           ;  argument for APRENB
                                       ;  ill-mem-ref only
       APRENB AC1     ,
       POPJ   TOPP    ,
```

4.3.5 APRINT (interrupt-routine)

Beim Auftreten eines Interrupts muessen wir pruefen, ob es ein ill-mem-ref-Fehler oder ein clock-flag-interrupt ist. Die Ursache des Interrupts steht im .JBCNI (126B). Im ersten Fall gehen wir sofort in den DEBUG-Modus ueber (mit Fehlermeldung und PC in AC0). Im anderen Fall testen wir, ob die interne Zeitgrenze ueberschritten worden ist. Wenn Ja, so erfolgt ein Sprung ins PASDDT, sonst wird das APRENB-UUO erneut aufgerufen und die Programmexekution an der unterbrochenen Stelle fortgesetzt.

Durch das Setzen der clock-flag werden alle Jiffies (1/50 Sekunden) ein Interrupt erzeugt. Wird dieses Programm gerade von der CPU bearbeitet, erfolgt ein Anspruch der Interruptroutine.

4.3.5.1 Code (fuer 4.3.5)
TEMP und LIMIT muessen als INTEGER-Groessen vereinbart werden

```
APRINT:MOVEM    AC0     ,TEMP
       HRRZ     AC0     ,.JBCNI
       TRNE     AC0     ,1000
       JRST     TIMINT
       MOVE     AC0     ,.JBTPC
       OUTSTR   [ASCIZ/
       ?        ILLEGAL MEMORY REFERENCE/]
       JRST     ERRDB1
TIMINT:SETZ     AC0     ,
       RUNTIM   AC0     ,
       SUB      AC0     ,LIMIT
       JUMPGE   AC0     ,TIMLIM
       MOVEI    AC0     ,21000
       APRENB   AC0     ,
       MOVE     AC0     ,TEMP
       JRSTF    @.JBTPC
TIMLIM:OUTSTR   [ASCIZ/
       ?        TIME LIMIT EXCEEDED/]
       MOVE     AC0     ,.JBTPC
       JRST     ERRDB1
```

4.4 Im PASDDT-Prosramm

4.4.1 Datenstrukturen

4.4.1.1 DYNENTRY
Der Record ist so zu veraendern, dass die Ausfuehrunssart des Prosramms (Batch oder Timesharing), welche in der linken Haelfte der Zelle 143B (s. 4.3.4) einsetrasen worden ist, abfrasbar wird.

```
DYNENTRY = PACKED RECORD
                DUMM1: BITS18;              (* LH 140B *)
                REGISTRS: ACR;              (* RH 140B *)
                STOPPY: INTEGER;            (*    141B *)
                DUMM2: BITS18;              (* LH 142B *)
                ENTRYPTR: ^DEBUGENTRY;      (* RH 142B *)
                DUMM3: BITS17;
                INTERACTIVE: BOOLEAN;       (* LH 143B *)
                STACKBOTTOM: ACR;           (* RH 143B *)
                STATUS: DEBUGSTATUS;        (*    144B *)
                TIME_LIMIT: INTEGER;        (*    145B *)
                (* vers1. Abschnitt 6.1 *)
           END;
```

4.4.2 Aenderunsen in den Prosrammstrukturen
Der Jetzise Prozedurkoerper von der PASDDT-Prozedur wird der Prozedurkoerper einer neuen Prozedur DEBUG_INTERACTIVE. Fuer den PMD fuehren wir eine neue Prozedur DEBUG_BATCH ein. Die Prozedur INIT wird in das neue "Hauptprosramm" uebernommen, das dann folsendermassen aussieht:

```
BEGIN
 INIT;
 IF ENTRY2.INTERACTIVE
 THEN
 DEBUG_INTERACTIVE
 ELSE
 DEBUG_BATCH;
END.
```

4.4.2.1 PROCEDURE SUCCBASIS

Um das in 2.2.2 erwaehnte Problem (Abbruch des Prozedur-Rueckverfolgungs-Algorithmus) zu loesen, pruefen wir ab, ob der Verweis (BASIS) auf die Ruecksprungadresse echt kleiner ist als der vorherige Verweis (OLDBASIS) :

```
PROCEDURE SUCCBASIS(SIDE: LEFTORRIGHT);
VAR
  OLDBASIS:ACR;
 BEGIN
  OLDBASIS:=BASIS;
  BASIS := ACRPOINT( BASIS^[0-1], SIDE );
  TESTGLOBALBASIS;
  IF ORD(OLDBASIS) <= ORD(BASIS)
  THEN
   BEGIN
    BASIS:=NULLPTR;
    NEWLINE;
    WRITE(TTY,'ERROR IN PROCEDURE-BACKTRACING');
   END;
 END (*SUCCBASIS*);
```

4.4.2.2 TRACEOUT

Vor dem Ausschreiben des Programmnamens sollte eine Abfrage eingefuehrt werden, ob LCP=NIL ist, d.h. kein Bezeichnerbaum eingetragen worden ist, was auftreten kann, wenn die Debug-Option lokal zurueckgesetzt wurde.

Ausserdem sollte fuer die Anzahl der maximal ausgegebenen Prozedurnamen in einer Zeile eine Variable eingefuehrt werden, welche mit 2 besetzt wird, falls es ein Timesharing-Job ist, andernfalls mit 5, da in dem einen Fall 3, und im anderen Fall 6 Prozedurnamen pro Zeile ausgegeben werden koennen.

4.4.2.3 PROCEDURE DEBUG_BATCH

```
PROCEDURE DEBUG_BATCH;
  BEGIN
    CASE ENTRY2.STATUS.KIND OF
          INITK: ;
      RUNTMERRK: TRACEOUT;
         OTHERS: WRITELN(TTY,'FEHLER IM DEBUG')
    END (*CASE*);
  END (* DEBUG_BATCH*);
```

5. Ergebnisse der Implementation

5.1 Reihenfolge der gesamten Ausgabe

Wegen der groesseren Zeilenlaenge sind wir dazu uebergegangen, den PASPMD nicht mehr in den LOG-File zu schreiben (vor jeder Zeile im LOG-File stehen 10 Zeichen), sondern auf einen besonderen File. Im Stapelbetrieb wird beim REWRITE-Statement auf diesen File als Device-Argument 'LPT' angegeben, so dass der PASPMD auch ohne Druckbefehl ausgegeben wird. Nach dem Ausschreiben einer Ueberschrift mit Angabe der Uhrzeit, des Datums, des Programm-Namens und der Dump-Version, wird die Zeilennummer, bei der die Programmexekution stoppte, und die beim Laufzeitfehler geltende dynamische Prozedurverschachtelung ausgegeben.

Anschliessend werden die lokalen Variablen der dynamischen Prozedurverschachtelungen bis zu einer maximalen Tiefe von 10 Prozeduren ausgegeben, wobei angegeben wird, ob diese Variablen zur Zeit gueltig sind. Sind die Variablen des Hauptprogrammes (globale Variable) noch nicht ausgegeben worden (d.h. es liegt eine dynamische Prozedurverschachtelung von mehr als 10 Prozeduren vor), werden ausserdem die lokalen Variablen der statischen Prozedurverschachtelung ausgehend von der unterbrochenen Prozedur ausgegeben, soweit die Ausgabe nicht schon erfolgte. Dann wird lediglich ein Verweis auf die schon ausgegebenen Variablen gegeben.

5.2 Reihenfolge der Ausgabe der Variablen einer Prozedur

Innerhalb einer Prozedur werden zuerst die skalaren Variablen (SCALAR, SUBRANGE, POINTER [die oktalen Adressen derselben]) und dann die strukturierten Variablen (SET, ARRAY, RECORD, FILE) jeweils alphabetisch geordnet ausgegeben.

Um eine bessere Ausnutzung des Papiers zu erreichen werden jetzt bis zu 4 skalare Werte in eine Zeile geschrieben. Deshalb haben wir die Prozedur NEWLINE veraendert. Sie beginnt jetzt in Abhaengigkeit der globalen Variablen TABS eine neue Zeile oder schreibt eine bestimmte Anzahl von Leerzeichen, damit die folgende Ausgabe in einer vorher von uns bestimmten Spalte beginnen kann. Die Tabulatorwerte (Spalten, bis zu denen Leerzeichen geschrieben werden sollen) speichern wir in dem Array TABULATOR ab. Ist der letzte Tabulatorwert ueberschritten worden, beginnen wir eine neue Zeile.

Im Augenblick unterteilen wir die Zeilen in 4 Spalten. Die oben beschriebene Methode des Tabulierens wird benutzt bei der Ausgabe von Variablen des Typs SCALAR, SUBRANGE und POINTER, sowie bei der Ausgabe von RECORD-Feldern und ARRAY-Komponenten mit den obengenannten Variablentypen. Ausserdem werden STRING-ARRAY's mit einer Zeichenanzahl, welche kleiner als 20 ist, tabuliert ausgegeben.

5.2.1 Variablen des Typs SCALAR, SUBRANGE und POINTER
Diese Klasse von Variablentypen geben wir mit Hilfe der Prozedur WRITESCALAR tabuliert aus. Dabei werden etwaige Bereichsueber- oder Unterschreitungen, sowie unzulaessige Besetzungen bei Zeiger- und Zeichen-Variablen kontrolliert und mit folgender Fehlermeldung kommentiert:

Ueberschreitung des Wertebereichs von Variablen mit selbstdefinierten Typen(wie bisher)
(out of range)

Ueber- oder Unterschreiten eines angegebenen Unterbereiches einer Variablen
(out of subrange)

Bei Zeiger-Variablen, deren Wert kleiner als NEWREG oder groesser als der Beginn des PASDDT-Speicherbereiches ist, nach dem Ausdrucken des Wertes die Meldung
(out of heap)

Sowie bei Zeichen-Variablen, deren Wert kleiner als 40 B (Leerstelle) oder groesser als 137 B (Pfeil nach links) ist
x(ill. Character)
wobei x der entsprechende ganzzahlige Wert ist.
(Diese Zeichen werden nicht ausgegeben, weil sie u. U. zu irritierenden Ausgaben fuehren.)

5.2.2. SET-Variablen
Jede Variable wird zu Beginn einer neuen Zeile geschrieben. Ansonsten erfolgt eine . Ausgabe wie beim bisherigen PASDDT-Programm.

5.2.3 ARRAY
ARRAY's, deren Komponenten vom Typ SCALAR, SUBRANGE, POINTER oder STRING (Laenge <20) sind, werden tabuliert ausgegeben, d.h. der Bezeichner wird am Anfang der ersten Zeile geschrieben und die Komponenten werden jeweils in der Form

[<Index>]=<Komponentenwert>

beginnend an der durch den Tabulator bestimmten Stelle ausgegeben (max 4 pro Zeile).
Bei mehrdimensionalen ARRAY's wird die Ausgabe um eine Stufe tiefer geschachtelt, d.h. an Stelle des Bezeichners der Index der zweiten Dimension (in eckigen Klammern) etc.
Um trotzdem noch etwas die Uebersicht behalten zu koennen, wird eine Art Blockstruktur (aehnlich wie im PASDDT) fuer tiefer geschachtelte Strukturen benutzt, d.h. sie werden etwas weiter eingerueckt.

Um Papierverschwendung zu vermeiden, wird gepruefft, ob aufeinander folgende Komponenten den selben Wert besitzen. In diesem Fall geben wir den ersten und den letzten Index der Indexfolge mit gleichen Komponentenwerten an, gefolgt von dem gemeinsamen Wert:

[<Index 1>]..[<Index n>]=<Komponentenwert>

Bei ARRAY's mit Komponenten des Typs SCALAR, SUBRANGE oder POINTER wird Byteweise auf Gleichheit gepruefft. Es wird der aktuelle Inhalt der Records GATTR festgehalten und die Funktion NEXTBYTE aufgerufen. Sie liefert als Ergebnis einen ganzzahligen Wert der naechsten Komponente. Unterscheidet sich dieser vom Wert der aktuellen Komponente, so wird GATTR zurueckgesetzt und der Wert der aktuellen Komponente ausgeschrieben, sonst wird die naechste Komponente gepruefft. Handelt es sich um hoeher strukturierte Komponenten, so wird maschinenwortweise verglichen.

5.2.4 RECORD

Die Ausgabe von RECORD's wurde dahingehend veraendert, dass die Feldliste mit Feldwerten bei den unter 5.2.1 genannten Typen tabuliert ausgegeben wird. Ist die Feldkomponente jedoch hoeher strukturiert, wird der Feldbezeichner eingerueckt in eine neue Zeile geschrieben.
Tagfields werden besonders gekennzeichnet und stehen immer in einer neuen Zeile.

5.2.5 FILE

Um dem Benutzer die Moeglichkeit zu geben, auch bei Laufzeitfehlern im Zusammenhang mit der FILE-Behandlung eine bessere Fehlersuche durchfuehren zu koennen, haben wir die uns wesentlich erscheinenden Informationen aus dem FILE-Verwaltungsblock entnommen und ausgedruckt.
Wenn der FILE noch nicht eroeffnet worden ist, welches wir daran erkennen, dass es keinen Verweis auf einen Puffer gibt (rechte und linke Haelfte von FILSTA + 2 = 0), geben wir die Meldung

File not opened

aus.
Sonst geben wir nach dem Filebezeichner den Geraetenamen, den Filenamen mit Extension, die PPN, den Protektion-Code und den Status (ASCII oder Binaer) aus.
In Abhaengigkeit von dem Eingabe / Ausgabe Modus und Status geben wir bei Eingabefiles den Wert des Praedikates EOF, sowie - wenn es sich um einen ASCII-File handelt - die Zeilennummer und den Wert des Praedikates EOLN aus. Bei einem Ausgabefile werden diese Angaben unterdrueckt. Als letztes wird noch der Inhalt der FILE-Puffer-Variablen ausgegeben.

Wir erhalten diese Informationen aus einem vom Compiler angelegtem und vom DECSystem-10 Betriebssystem bzw. der PASCAL-Laufzeitunterstuetzung gesetzten 'Fileblock' (siehe [9]), durch Zugriff auf die Felder einer strukturierten Variablen vom Typ RECORD, die genau der Definition eines Fileblocks entspricht.

5.3 Ausgabe des Haldeninhalts

Die zunaechst geplante Ausgabe des Haldeninhaltes ist in 2.2.4 ausfuehrlich beschrieben. An der Verwirklichung hat sich gegenueber unserem Vorschlag nichts geaendert.
Die Ausgabe sieht folgendermassen aus:

> Jede allozierte Variable beginnt in einer neuen Zeile, an deren Anfang die oktale Adresse (entsprechend einem Zeiger auf diese Variable) steht. Die Ausgabe beginnt bei dem kleinsten zulaessigen Wert eines Zeigers (beim Inhalt von NEWREG). Ansonsten erfolgt die Formatierung wie bei Variablen des entsprechenden Typs (siehe oben).

Bei der Ausgabe des Haldeninhalts koennen folgende Meldungen ausgegeben werden:

> Gibt es zu der allozierten Variablen keinen Verweis auf den zugehoerigen STRUCTURE-Record, so wird die oktale Adresse ausgegeben, gefolgt von
>
> ```
> type of variable not known
> ```
>
> Werden Fehler in der Verweiskette bemerkt, welche durch Ueberschreiben der Verweisworte verursacht sein koennen, wird der Haldendump abgebrochen und ausgegeben
>
> ```
> can't continue the heapdump
> ```

5.4 Weitere Aenderungen

Durch die Ausgabe des PASPMD auf einen besonderen File war es uns leicht moeglich, die DUMP-Erweiterungen auch den interaktiven Benutzern des PASDDT zur Verfuegung zu stellen. Danach ist es jetzt moeglich, auch von der Benutzerkonsole eine Ausgabe des Inhaltes der Halde, bzw. des Laufzeitkellers nach den Konventionen des PASPMD's auf einen Datensatz (XXXXXX.PMD, X in ['0'..'9']) zu erzwingen.

6. Schwierigkeiten und Erfahrungen bei der Implementation

6.1. Aenderungen in der PASDDT-Laufzeit-Unterstuetzung

Die Aenderungen, welche im Abschnitt 4 schon beschrieben worden sind, konnten ohne groessere Probleme durchgefuehrt werden.
Fuer die interne Zeitbeschraenkung mussten wir uns eine weitere Speicherzelle 145B (oktal) innerhalb des DEBUGENTRY-Records reservieren.
Die Berechnung des internen Zeitlimits wurde aus der PASDDT-Laufzeitunterstuetzung in die DEBUG_BATCH Prozedur verlegt.

6.2. Aenderungen in der PASCAL-Laufzeitunterstuetzung

Zusaetzlich zu den im Abschnitt 4 angekuendigten Aenderungen wurde folgendes implementiert:

> Beim Auftreten eines Kellerueberlaufs wird gepruefft, ob sich das Programm in der Initialisierungsphase befindet. Ist dies nicht der Fall, wird die Adresse der Prozeduraktivierung berechnet und BASIS dann zurueckgesetzt, ansonsten wird der Programmlauf beendet.

Da der Batch-Controller bei der Ausgabe eines Fragezeichens in der ersten Spalte des Kontroll-Files (TTY-File) in den ^C-Status uebergeht und dann einen System-Dump startet, beginnen wir alle Fehlermeldungen der Laufzeitunterstuetzung mit einem "%?", statt wie bisher mit einem "?".

6.3. Aenderungen im PASDDT-Programm

Zuerst haben wir die im Abschnitt 4 erwaehnten Aenderungen durchgefuehrt, welche eine Prozedur-Rueckverfolgung ermoeglichen (TRACEOUT, SUCCBASIC, DEBUG_BATCH, DEBUG_INTERACTIVE).
Anschliessend wandten wir uns der Ausgabe der Variablen zu:
Anfangs versuchten wir durch Zwischenpufferung und spaltenweise Ausgabe (bis zu 6 Spalten pro Druckseite) eine maximale Ausnutzung des Papiers zu erreichen. Dabei stellte sich heraus, dass diese Ausgabeform besonders bei strukturierten Variablen zur Unuebersichtlichkeit fuehrte.
Danach entschlossen wir uns fuer eine Ausgabeform, die zwar nicht so kompakt, aber dafuer uebersichtlich ist (siehe obige Beschreibung).

6.4 Aufgetretene und beseitigte Fehler

a) Beim Austesten des Programmes ergab sich folgender Fehler im PASDDT-System:
Beim Ausfuehren einer REWRITE-Anweisung, bei der neue Puffer anzulegen waren, wurde der Keller des PASDDT-Speicherbereiches ueberschrieben. Der Grund war, dass die Speicherzelle .JBFF in der JOB-DATA-AREA nach dem CORE-UUO in der PASDDT-Laufzeitunterstuetzung nicht auf den neuen Wert von .JBREL gesetzt wurde.
Es wundert uns, weshalb dieser Fehler erst jetzt bemerkt worden ist.

b) Bei einer versuchsweisen Selbstcompilation stellten wir fest, dass SET's fehlerhaft compiliert wurden, wenn innerhalb eines SET's eine Variable und eine Konstante angegeben wurden.

Literaturhinweise
=================

[0] H.H. Nasel et al.
Erweiterung von Sprachdefinition, Compiler und Laufzeitunterstuetzung bei PASCAL
Institutsmitteilungen Nr. 16
Institut fuer Informatik, Hamburg 1975

[1] Advanced Course on Software Engineering
F.L. Bauer (ed.), Lecture Notes in Economics and Mathematical Systems, vol. 81
Springer Verlag, Berlin-Heidelberg-New York 1973

[2] K. Jensen / N. Wirth
PASCAL User Manual and Report
Lecture Notes in Computing vol. 18
Springer Verlag, Berlin-Heidelberg-New York 1974

[3] Peter Putfarken
Ein interaktives Debugsystem fuer PASCAL
Diplomarbeit, Institut fuer Informatik, Hamburg 1976

[4] F.W. Lorenz, P.-J. Stirl
Uebertragung eines PASCAL-Compilers auf das DECSystem-10
Diplomarbeit, Institut fuer Informatik, Hamburg 1974

[5] PASCAL.HLP
Dokumentation des DECSystem-10 PASCAL-Dialektes
Datensatz auf der PDP10, Institut fuer Informatik, Hamburg 1975

[6] DECSystem-10, Assembly Language Handbook
Digital Equipment Corporation
Maynard Mass. 1972

[7] DECSystem-10, User's Handbook
Digital Equipment Corporation
Maynard Mass. 1972

[8] E.H. Satterthwaite,Jr.
Debugging tools for high level languages
University of Newcastle upon Tyne
Computing Laboratory
Technical Report Series No. 29, 1971

[9] C.-O. Grosse-Lindemann
Diplomarbeit, Institut fuer Informatik, Hamburg 1975
und
C.-O. Grosse-Lindemann, H.-H. Nasel
Postlude to a PASCAL-compiler bootstrap on a DECSystem-10
Software-Practice and Experience, vol. 6, 29-42, 1976

Waehrend der Analysephase des Projektes wurden die folgenden Quellprogramm-Fassungen verwendet (waehrend der Ueberarbeitung des vorliegenden Berichtes sind alle hier beschriebenen Erweiterungen in die neue PASCAL-Compilerversion vom Juni 1976 eingearbeitet worden).

[10] PASREL.PAS vom 2.September 1975

[11] RUNSPN.MAC vom 2.September 1975

[12] DEBSPN.MAC vom 7.Juli 1975

[13] DEBUG.PAS vom 7.Juli 1975

Bisher veröffentlichte Instituts-Mitteilungen:

1 JESSEN, E.
RECHNERORGANISATION
III/73, 156 P

2 BROCKMANN, H.-H., FRIESLAND, G., HABERMANN, H.-J., LORENZ, P.-W.,
NAGEL, H.-H., SENGLER, H.-E., STIRL, P. J.
EIN PASCAL-COMPILER FUER DIE PDP-10
V/73, 34 P

3 NAGEL, H.-H.
ON NETWORK LANGUAGES
/73, 16 P

4 ULLRICH, G.
EIN MULTITASKING-SUPERVISOR FUER DIE PDP-11
I/73, 29 P

5 FRIESLAND, G., GROSSE-LINDEMANN, C.-O., LORENZ, P.-W., NAGEL, H.-H.,
STIRL, P. J.
A PASCAL COMPILER BOOTSTRAPPED ON A DEC-SYSTEM 10
II/73, 14 P

6 SCHWENKEL, F.
ARBEITSBLAETTER ZUR IMPLEMENTATION DER PROGRAMMIERSPRACHE EULER
XI/73, 12 P

7 JESSEN, E.
ARBEITSPROGRAMM DER FORSCHUNGSGRUPPE ENTWURFSLEHRE FUER MODULARE
SYSTEME
/74, 14 P

8 FICHTEL, H., FISCHER, H., LEHMANN, H., LINDE, H., SCHMIDT, J.,
SCHWENKEL, F.
MIST (MINIMAL-SPRACHEN-EXPERIMENT)
V/74, 19 P

9 KUPKA, I., OBERQUELLE, H., WILSING, N.
PROJEKT EINER RAHMENDIALOGSPRACHE-RDS
X/74, 10 P

10 BOLEY, H.
EINFACHE NATUERLICHSPRACHLICHE DIALOGE MIT EINEM SEMANTISCHEN
NETZWERK
XI/74, 87 P

11 SCHEFE, P.
BESCHREIBUNG EINES PROGRAMMS FUER DAS KONSTRUIEREN UND TESTEN VON
FORMALEN GRAMMATIKEN IM DIALOG MIT EINER RECHENANLAGE
I/75, 42 P

12 OBERQUELLE, H.
DIALOGSYSTEME AUS DER SICHT DES BENUTZERS =
GRUNDLAGEN FUER EINE TRANSPARANTE BESCHREIBUNG UND BEGRIFFSBILDUNG
III/75, 13 P

13 MEYER, A.

EINFUEHRUNG IN DAS RETRIEVALSYSTEM DER INFORMATIKBIBLIOTHEK HAMBURG
II/75, 17 P

14 WENDT, S.
FUNKTIONSBESCHREIBUNG DES INTEGRIERTEN PROZESSORBAUSTEINS INTEL 8ØØ
III/75, 2Ø P

15 ULLRICH, G.
MDOS-11, EIN MULTITASKING DISK OPERATING SYSTEM FUER DIE PDP 11
II/75, 43 P

16 NAGEL, H.-H. (ED.)
ERWEITERUNG VON SPRACHDEFINTION, COMPILER- UND LAUFZEITUNTERSTUETZUNG
BEI PASCAL,
ERFAHRUNGEN UND ERGEBNISSE EINES PRAKTIKUMSPROJEKTES IM
INFORMATIK-GRUNDSTUDIUM,
V/75, 39 P

17 ALBRECHT, H., FISCHER, B., PLATZ, D.
ANWENDUNG SPEZIELLER BILDBEARBEITUNGSVERFAHREN ZUR ERKENNUNG VON
ORGANGRENZEN UND TUMOREN IN LOKALISATIONSDIAGNOSTISCHEN
LEBERSZINTIGRAMMEN IN EINEM INTERPRETATIVEN DIALOGSYSTEM,
VI/75, 74 P

18 WEICKER, R.
ZUR SITUATION DER INFORMATIK-STUDIENANFAENGER
VI/75, 18 P

19 WITTIG, T.
LISP 1.6 II - EINE ERWEITERUNG DER STANFORD-VERSION
XI/75, 25 P

2Ø HEIBEY, H.-W., LUTTERBECK, B., ROHLFS, S., SCHUELER, U.
AUSWIRKUNGEN DER DATENVERARBEITUNG AUF DEN EINZELNEN ANWENDER
UND ORGANISATIONEN, DIE DIE DATENVERARBEITUNG ANWENDEN,
XI/75, 146 P

21 NAGEL, H.-H.
PASCAL FOR THE DECSYSTEM-1Ø, EXPERIENCE AND FURTHER PLANS,
XI/75, 7 P

22 HUELLER, F., WENDT, S.
DATALINE-KANALWERK FUER DIE PDP-11
XI/75, 38 P

23 BRUNNSTEIN, K., RUDOLPH, A., SACHSE, H., UHLENHAUT, G.
PASCAL-E, EINE EINFUEHRUNG IN EINEN SCHULORIENTIERTEN
(ELEMENTAREN) PASCAL-SUBSET
XI/75, 43 P

24 BRUNNSTEIN, K. (ED.)
ZUR KONZEPTION EINES FRAGE-ANTWORT-SYSTEMS FUER DEN
COMPUTER-GESTUETZTEN UNTERRICHT
XII/75, 74 P

25 BORN, H. U., KUEHL, G., STEIMER, D.
EVALUATION NETWORKS ALS BESCHREIBUNGSMITTEL FUER RECHENSYSTEME
I/76, 2Ø9 P

26 DOERING, D., EHMER, R.
ENTWURF UND IMPLEMENTIERUNG EINES AUF DEM TISCHRECHNERKONZEPT AUFBAUENDEN KERNS FUER EINE EXPERIMENTELLE DIALOGSPRACHE
I/76, 41 P

27 HEIBEY, H. W., LUTTERBECK, B., ROHLFS, S., TOEPEL, M.
THEORETISCHE BETRACHTUNGEN ZU DEN AUSWIRKUNGEN DER DATENVERARBEITUNG
III/76, 106 P

28 SCHMIDT, J. W.
UNTERSUCHUNG EINER ERWEITERUNG VON PASCAL ZU EINER DATENBANKSPRACHE
III/76,

29 NAGEL, H.-H.
DISKUSSION DES PROZESSBEGRIFFS
POSTSKRIPTUM ZU DEM VON H.-H.NAGEL VORGETRAGENEN TEIL DER VORLESUNG BETRIEBSSYSTEME IM SS 1975, AUSGEARBEITET VON H. ECKHARDT, K. GERHARDT, C. RUHE
IV/76, 98 P

30 BEHRENS, M., FRIESLAND, G.
BENUTZUNGSANLEITUNG FUER DAS COMPILER-WRITING-SYSTEM MONTREAL - ANGEPASST AN DAS DEC-SYSTEM 10
IV/76,

31 ULLRICH, G.
SEQUENTIAL PASCAL FUER DIE PDP-11 UNTER DEM BETRIEBSSYSTEM DOS-11
XII/75, 28 P

32 FICHTEL, H.
ZUR UEBERTRAGUNG VON CONCURRENT UND SEQUENTIAL PASCAL AUF DIE INTERDATA M85
V/76,

33 BUDY, E., DRESCHLER, L.
UNTERSUCHUNG ZUR IDENTIFIKATION EINES BEWEGTEN OBJEKTES (PKW) IN DER VIEDEO-BILDSEQUENZ EINER STRASSENSZENE
V/76, 51 P