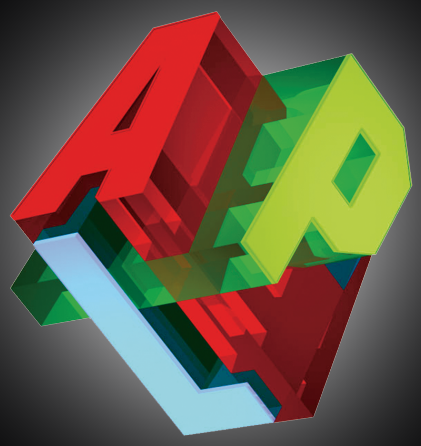
Heft 1/2011 Jahrgang 30 ISSN - 1438-4531 APL-Germany e.V. RHOMBOS-VERLAG

APL-Journal

A Programming Language



Frank Schmidt-Kelletat

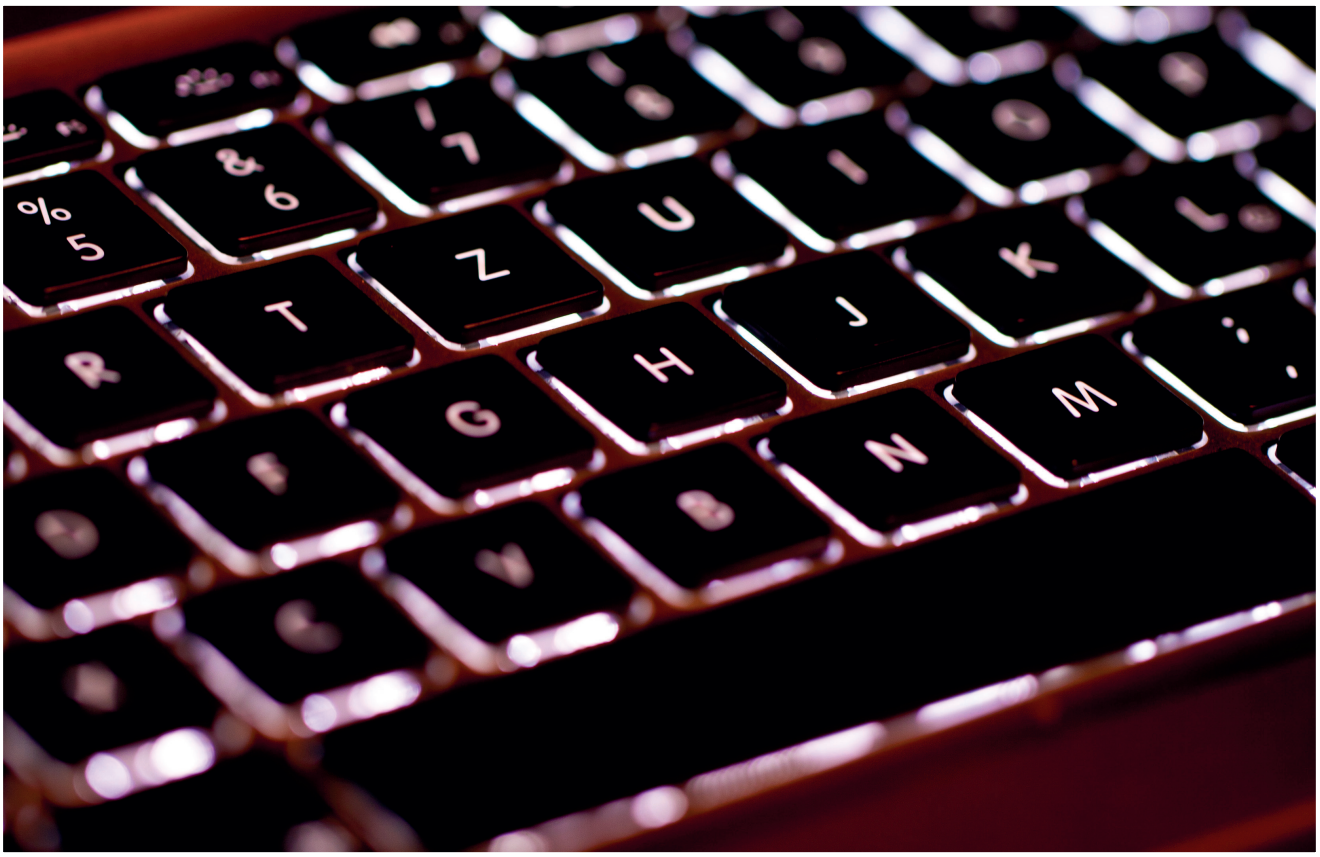
Datensatzkomprimierung mit IBM-APL2

Reiner Nussbaum

unix4APLers - unix für Erwachsene

Rolf Herminghaus

Fehlersuche mit APL+WIN 9.1



APL-Journal

30. Jg. 2011, ISSN 1438-4531

Herausgeber: Dr. Reiner Nussbaum, APL-Germany e.V., Mannheim, http://www.apl-germany.de

Redaktion: Dipl.-Volksw. Martin Barghoorn (verantw.), TU Berlin, Franklinstr. 28, D-10587 Berlin, Tel. (030) 314 24392, Fax (030) 314 25901

Verlag: RHOMBOS-VERLAG, Berlin, Kurfürstenstr. 17, D-10785 Berlin, Tel. (030) 261 9461, Fax (030) 261 6300, eMail: verlag@rhombos.de, Internet: www.rhombos.de

Erscheinungsweise: halbjährlich

Erscheinungsort: Berlin

Satz: Rhombos-Verlag, Vicky Kletke

Druck: dbusiness.de GmbH, Berlin

Copyright: APL Germany e.V. (für alle Beiträge, die als Erstveröffentlichung erscheinen)

Fotonachweis Titelseite: Martin Müller / pixelio.de

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutzgesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürfen. Eine Haftung für die Richtigkeit der veröffentlichten Informationen kann trotz sorgfältiger Prüfung von Herausgeber und Verlag nicht übernommen werden. Mit Namen gekennzeichnete Artikel geben nicht unbedingt die Meinung des Herausgebers oder der Redaktion wieder. Für unverlangte Einsendungen wird keine Haftung übernommen. Nachdruck ist nur mit Zustimmung des Herausgebers sowie mit Quellenangabe und Einsendung eines Beleges gestattet. Überarbeitungen eingesandter Manuskripte liegen im Ermessen der Redaktion.

Liebe APL Freunde,

ich freue mich, Ihnen das neue APL-Journal zukommen zu lassen.

Im letzten Jahr hatten wir in Berlin die internationale APL2010 Konferenz ausgerichtet. Die Beiträge davon haben wir in einem separaten APL Journal als Konferez Proceedings herausgegeben.

In diesem Frühjahr haben wir die APL Germany Tagung und die Tagung der IBM GSE APL workgroup bei der Ergo Versicherung abgehalten. An dieser Stelle möchte ich mich bei unseren Gastgebern noch einmal für die freundlichen Aufnahme bedanken.

Auf der Veranstaltung haben wir interessante Einblicke in die Nutzung von APL aus der Anwendersicht erhalten. Ebenso interessant sind die Beiträge der Hersteller von APL Systemen über deren Erfolge bei der Weiterentwicklung gewesen.

In diesem APL-Journal sind die bisher vorliegenden aktuellen Beiträge abgedruckt.

Ich wünsche Ihnen viel Freude bei dem Lesen der vorliegenden Ausgabe des APL-Journals.

Ihr

Reiner Nussbaum

p.s.: Natürlich sind wir immer auf der Suche nach interessanten Vorträgen und Artikeln für das APL-Journal. Daher der Aufruf an alle Interessierten: Wenn Sie etwas interessantes beitragen möchten, dann wenden Sie sich bitte an mich oder an Martin Barghoorn.

Frank Schmidt-Kelletat
Datensatzkomprimierung mit IBM-APL2
Angenommen, Sie halten Daten in strukturierter Form vor, und Sie haben den Wunsch, nur einen Teil davon für weitere Analysen zu extrahieren, so bedarf es eines Werkzeugs, dass dem Anwender ermöglicht, die gesuchten Datensätze über die Formulierung von Filterbedingungen einzugrenzen, als auch die Felder bedarfsgerecht auszuwählen.

Reiner Nussbaum
unix4APLers – unix für Erwachsene
Im Frühjahr 2008 hat der Autor bei sourceforge.net das Projekt unix4APLers begonnen. Der Funktionsumfang der unix shells und auch der Windows dos-box ist noch relativ gering, andererseits ergibt sich dadurch eine kompakte, effiziente shell Umgebung. Ziel des Projektes ist, für umfangreichere Aufgabenstellungen ein geeignetes Arbeitsumfeld bereitzustellen.

Rolf Herminghaus
Fehlersuche mit APL+WIN 9.1
Der Artikel behandelt einige Features der Sprache APL und darauf basierende einfache Hilfsprogramme, die die Fehleranalyse in komplexen Software-Systemen deutlich erleichtern können.

Inhalt

Seite

Datensatzkomprimierung mit IBM-APL2	**5**
unix4APLers – unix für Erwachsene	**10**
Fehlersuche mit APL+WIN 9.1	**23**
Impressum	**34**

Frank Schmidt-Kelletat

Datensatzkomprimierung mit IBM-APL2

Angenommen, Sie halten Daten in strukturierter Form vor, und Sie haben den Wunsch, nur einen Teil davon für weitere Analysen zu extrahieren, so bedarf es eines Werkzeugs, dass dem Anwender ermöglicht, die gesuchten Datensätze über die Formulierung von Filterbedingungen einzugrenzen, als auch die Felder bedarfsgerecht auszuwählen. Durch beide Varianten wird die Ausgangsdatei entweder vertikal oder horizontal auf die relevanten Informationen verjüngt bzw. komprimiert. Ist darüber hinaus noch eine Verdichtung der verbleibenden Datensätze gewünscht, so erfährt die Komprimierung eine zusätzliche Eskalationsstufe.

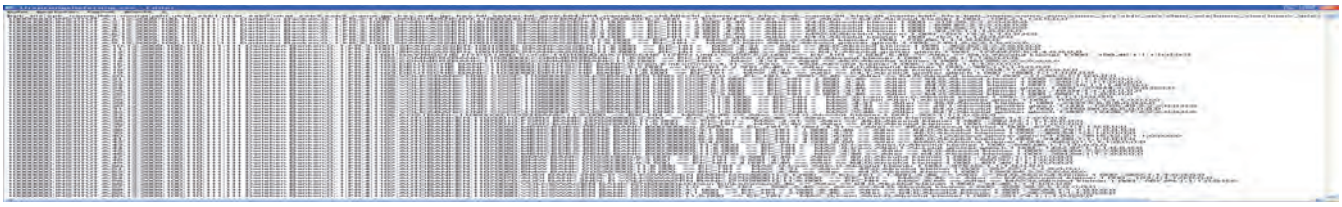


Abb. 1: Ursprungsdatei.CSV – enthält 31 Spalten und ca. 2,7 Mio. Sätze (Größe: ca. 660 MB)

Das Datenaufkommen hinsichtlich unterschiedlicher betriebswirtschaftlicher Problemstellungen variiert und kann für konkrete Analysen überbordend sein. Geht man von dem Fall aus, dass eine Datenlieferung in Form einer CSV-Datei vorliegt, die wesentlich mehr Spalten enthält als benötigt werden, und Datensätze, die statt eines zu untersuchenden Sparkassen-Instituts mehrere beinhaltet, bedarf es einer DV-technischen Möglichkeit, zielgerichtet Informationen zu extrahieren.

Entscheidet man sich bei einer Entwicklungslösung für das IBM-APL2 als Basissoftware, liegt auf den ersten Blick auf der Hand, dass der Workspace des APL2 die natürliche Barriere bei der Bearbeitung der CSV-Datei bildet. Mit Hilfe des 12er-Prozessors bietet APL aber eine Variante, Dateien in den lesenden oder schreibenden Zugriff zu nehmen, die weit größer sind als der reservierte Speicher der APL-Sitzung an MB zur Verfügung stellt, so dass der Workspace letztendlich kein K.o.-Kriterium für die

Entwicklung eines allgemeingültigen Werkzeuges darstellt. Hierbei muss natürlich noch ein weiterer Aspekt mit einbezogen werden: die Selektionen müssen performant sein. Da die Selektionen auf Felder wirken, liegt es nahe, die satzorientierte Ausgangsdatei zu invertieren – man erhält soviele Vektoren wie die Ausgangsdatei Spalten hat, mit sovielen Attributen wie die Ausgangsdatei an Sätzen enthält.

In einem ersten Schritt ist also dafür zu sorgen, dass unter Berücksichtigung des limitierten Speicherplatzes, die Ausgangsdatei umgewandelt wird. Diese invertierte Datei bildet dann die Grundlage für alle nachfolgenden konkreten Selektionen.

Technisch wird die Ausgangsdatei mit der folgenden Syntax in den Zugriff genommen:

```
(TFR DATEI TG0 1 * C1 2 50000 * )12   NA TV_
```

Hierdurch wird ein Vektor von Matrixen erzeugt, wobei jede Matrix z.B. 50000 Datensätze enthält, die Ausgangsspalten sind dadurch zwar zerschlagen, aber für die weitere Verarbeitung keinesfalls zerstört. Auf dem Weg zur Erzeugung der gewünschten spaltenorietierten Datei, kann man sich nämlich nun das Wissen zu Nutze machen, dass Spalten im CSV-Format (üblicherweise) durch Semikola getrennt werden. Somit lässt sich in einem Loop, jede Spalte dadurch rekonstruieren, indem man in jeder Matrix die gewünschte Teilspalte ausfiltert und letztlich alle so gewonnenen Teilspalten durch Verketten wieder zur gesamten Ausgangsspalte zusammensetzt. Mit Hilfe des 12er-Prozessors wird die Spalte als Vektor in eine Zwischendatei geschrieben. Aus Performancegründen werden darüber hinaus

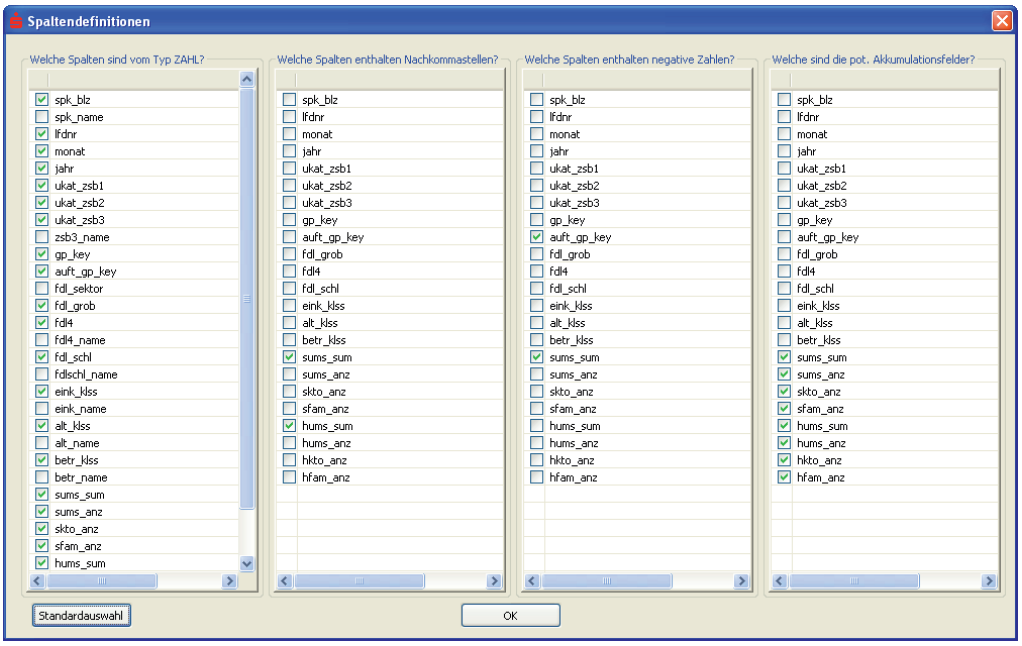


Abb. 2: Definition der Felder

Metadaten zu dieser Spalte, wie z. B. „Anzahl der eindeutigen Attribute" oder „Ausprägung der eindeutigen Attribute", ebenfalls vermerkt.

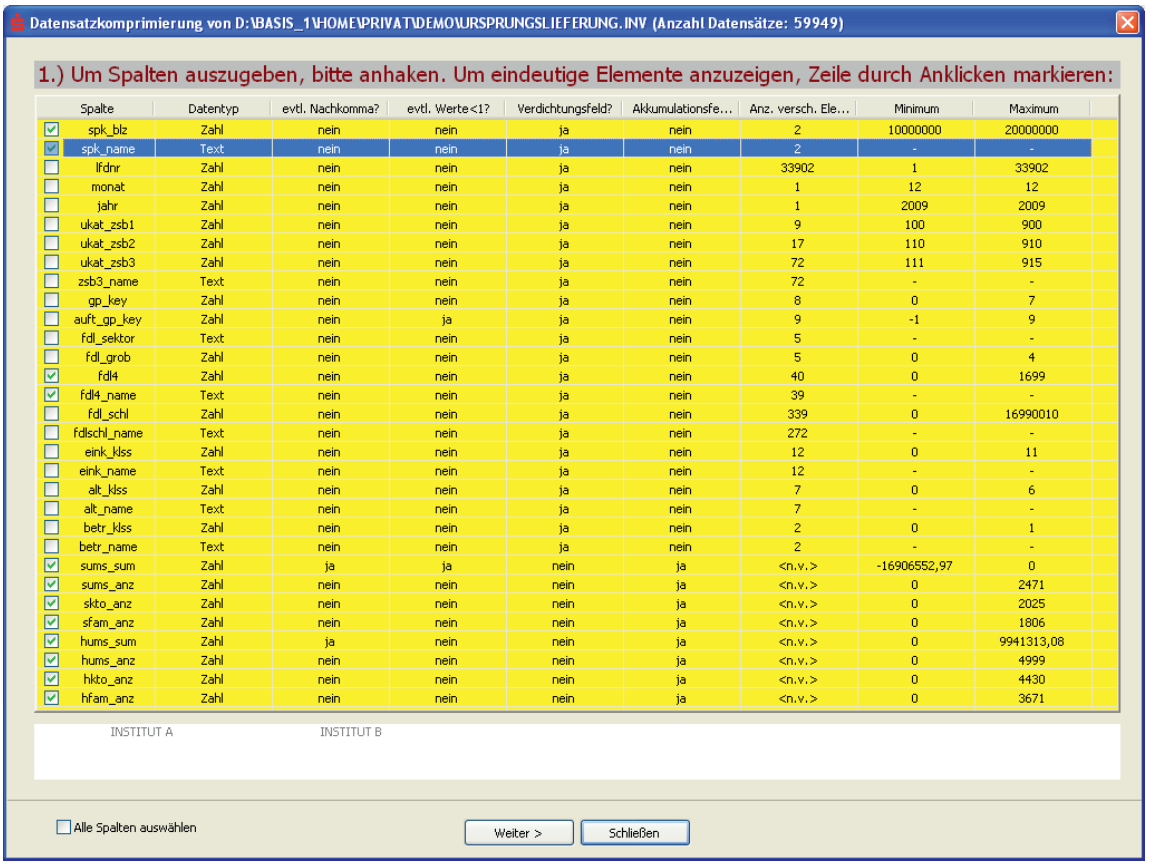
Nach diesem Verfahren werden auch alle weiteren Spalten gefiltert, verkettet und analysiert, so dass abschließend eine Datei verfügbar ist, die zwar ihre Struktur verändert, dabei aber keine Information verloren, sondern ganz im Gegenteil, an relevanter Informationsvielfalt gewonnen hat.

Mit der spaltenorientierten Datei steht nun ein optimales Konstrukt zur Verfügung, das über eine Benutzeroberfläche *intuitiv* und *leistungsstark* auswertbar ist.

Die Dialogsteuerung ist in drei Schritte unterteilt:

- Auswahl der auszugebenden Felder
- Filterung der Datensätze
- Verdichtung und Ausgabe der Ergebnisdatensätze

Im ersten Schritt (s. Abbildung 3) erkennt der Benutzer alle Felder aus der CSV-Ausgangsdatei wieder, angereichert um entsprechende Metainformationen. Die Information, welche eindeutigen Attribute pro Feld vorliegen, lässt sich durch Markieren der jeweiligen Zeile gewinnen – die Elemente werden



Spalte	Datentyp	evtl. Nachkomma?	evtl. Werte<1?	Verdichtungsfeld?	Akkumulationsfe...	Anz. versch. Ele...	Minimum	Maximum
spk_blz	Zahl	nein	nein	ja	nein	2	10000000	20000000
spk_name	Text	nein	nein	ja	nein	2	-	-
lfdnr	Zahl	nein	nein	ja	nein	33902	1	33902
monat	Zahl	nein	nein	ja	nein	1	12	12
jahr	Zahl	nein	nein	ja	nein	1	2009	2009
ukat_zsb1	Zahl	nein	nein	ja	nein	9	100	900
ukat_zsb2	Zahl	nein	nein	ja	nein	17	110	910
ukat_zsb3	Zahl	nein	nein	ja	nein	72	111	915
zsb3_name	Text	nein	nein	ja	nein	72	-	-
gp_key	Zahl	nein	nein	ja	nein	8	0	7
auft_gp_key	Zahl	nein	ja	ja	nein	9	-1	9
fdl_sektor	Text	nein	nein	ja	nein	5	-	-
fdl_grob	Zahl	nein	nein	ja	nein	5	0	4
fdl4	Zahl	nein	nein	ja	nein	40	0	1699
fdl4_name	Text	nein	nein	ja	nein	39	-	-
fdl_schl	Zahl	nein	nein	ja	nein	339	0	16990010
fdlschl_name	Text	nein	nein	ja	nein	272	-	-
eink_klss	Zahl	nein	nein	ja	nein	12	0	11
eink_name	Text	nein	nein	ja	nein	12	-	-
alt_klss	Zahl	nein	nein	ja	nein	7	0	6
alt_name	Text	nein	nein	ja	nein	7	-	-
betr_klss	Zahl	nein	nein	ja	nein	2	0	1
betr_name	Text	nein	nein	ja	nein	2	-	-
sums_sum	Zahl	ja	ja	nein	ja	<n.v.>	-16906552,97	0
sums_anz	Zahl	nein	nein	nein	ja	<n.v.>	0	2471
skto_anz	Zahl	nein	nein	nein	ja	<n.v.>	0	2025
sfam_anz	Zahl	nein	nein	nein	ja	<n.v.>	0	1806
hums_sum	Zahl	ja	nein	nein	ja	<n.v.>	0	9941313,08
hums_anz	Zahl	nein	nein	nein	ja	<n.v.>	0	4999
hkto_anz	Zahl	nein	nein	nein	ja	<n.v.>	0	4430
hfam_anz	Zahl	nein	nein	nein	ja	<n.v.>	0	3671

Abb. 3: Auswertungsübersicht

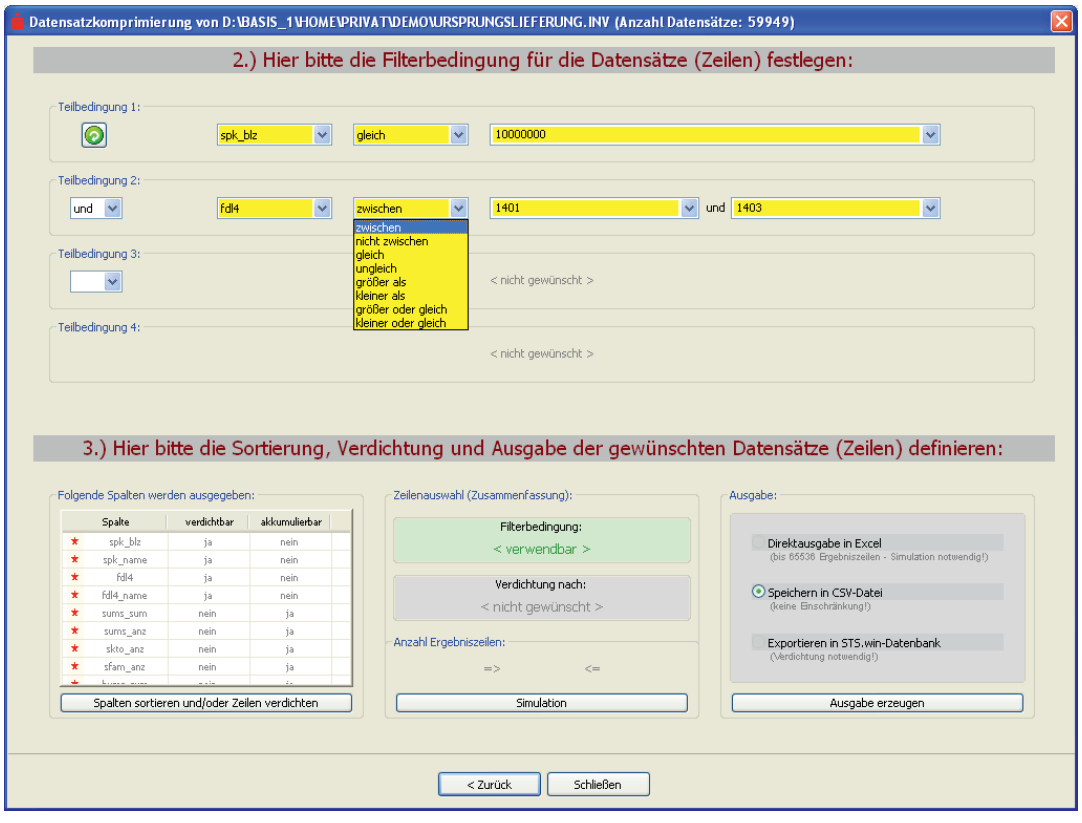


Abb. 4: Dialog für Filterbedingungen etc.

am unteren Bildrand angezeigt. Die Auswahl, welche Felder letztendlich ausgegeben werden sollen, nimmt der User über die Check-Boxes vor. Mit der Auswahl der Felder ist aber noch nichts darüber gesagt, welche Datensätze ausgegeben werden sollen. Ohne eine genaue Spezifizierung würden uneingeschränkt alle Datensätze über die gewählten Felder als Output generiert.

Wie in der Abbildung oben demonstriert, lassen sich die gesuchten Datensätze im zweiten Schritt durch die Formulierung einer Filterbedingung spezifizieren. Dazu stehen die Verknüpfungs- sowie die üblichen Vergleichsoperatoren zur Verfügung. Für die Vergleichsoperationen werden jeweils die Felder mit ihren eindeutigen Attributen zur Auswahl angeboten. Hierbei schlägt aus Performance-Sicht positiv zu Buche, dass diese Elemente während des Umwandlungsprozesses schon ermittelt und gespeichert wurden, so dass hier nur noch die Zuweisung auf die Combo-Box geschehen muss. Der Dialog ist selbstverständlich so konzipiert, dass er ab der dritten Teilbedingung das Setzen von Klammern ermöglicht. Der Benutzer hat nun die notwendigen Selektionen vorgenommen, um die Datensätze mit den relevanten Feldern ausgeben zu können. Der abschließende dritte Schritt stellt dafür verschiedene Output-Varianten zur Verfügung, zusätzlich ist es hier aber noch möglich, Verdichtungskriterien zu bestimmen und die Reihenfolge der Felder für die Ausgabe zu verändern.

Standardmäßig ist der Radio Button für die Ausgabe in eine CSV-Datei aktiviert. Quittiert der Anwender diese Vorauswahl durch Betätigen des Buttons „Ausgabe erzeugen", so erscheint ein Dialogfenster, das die Eingabe des gewünschten Dateinamens und die Auswahl des gewünschten Zielordners ermöglicht.

Möchte der Benutzer die Trefferzeilen seiner Selektion direkt in Excel anzeigen, macht das nur Sinn, wenn die Anzahl der Trefferzeilen 65536 (max. Anzahl an Zeilen in Excel 2003) nicht übersteigt - diese Information erhält man durch Betätigen des Buttons „Simulation". Eine dritte Variante, um Datenzeilen zu behandeln, ist die Möglichkeit der Speicherung in einer OLAP-Datenbank über „Exportieren in STS.win-Datenbank". Diese Ausgabespielart ist deswegen interessant, weil sich hierdurch Zeitreihen aufbauen lassen, die weitere, vertiefende Analysen in Subsystemen etc. ermöglichen. Voraussetzung ist hierbei allerdings, dass die selektierten Datenzeilen verdichtet werden. Über den Button „Spalten sortieren und/oder Zeilen verdichten" öffnet sich folgender Dialog:

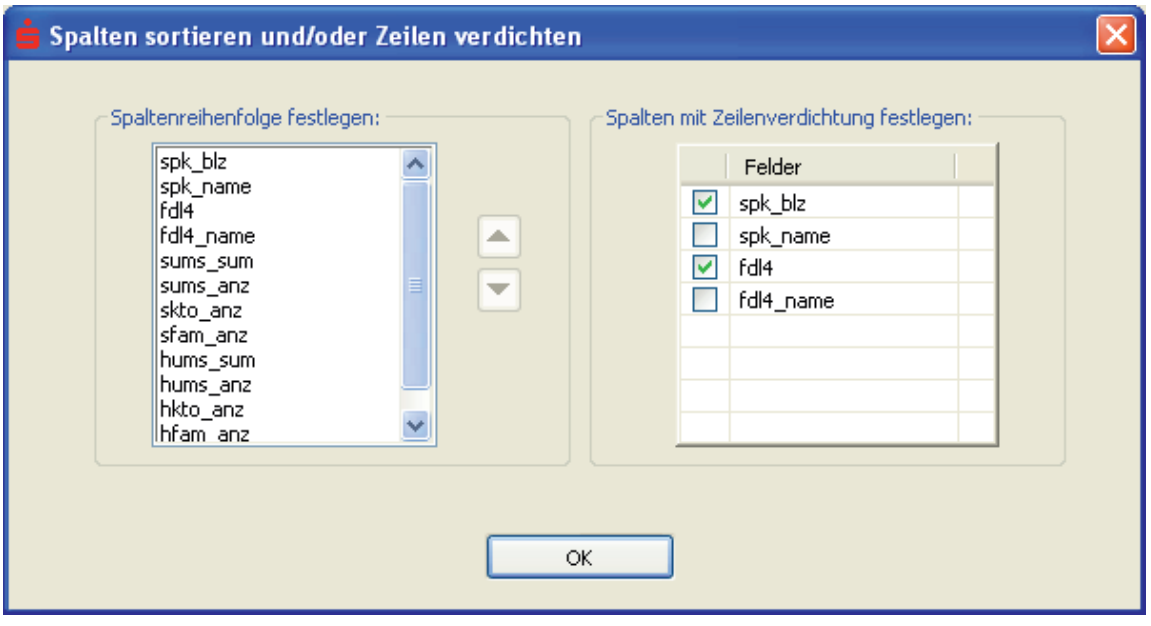


Abb. 5: Sortier- und Verdichtungsdialog

Über den Autor:

Frank Schmidt-Kelletat ist als Diplom-Kaufmann der Wirtschaftsinformatik beim Rheinischen Sparkassen- und Giroverband tätig.

E-Mail: frank.schmidt-kelletat@rsgv.de

Reiner Nussbaum

unix4APLers – unix für Erwachsene

Im Frühjahr 2008 hat der Autor bei sourceforge.net das Projekt unix4APLers begonnen. Die Software ist unter GPL frei verfügbar (http://sourceforge.net/projects/unix4aplers/). Der Funktionsumfang der unix shells und auch der Windows dos-box ist noch relativ gering, andererseits ergibt sich dadurch eine kompakte, effiziente shell Umgebung. Ziel des Projektes ist, für umfangreichere Aufgabenstellungen ein geeignetes Arbeitsumfeld bereitzustellen. Die Erweiterungen können auch gruppiert in Bausteinen implementiert werden. Große Anstrengungen wurden unternommen, damit die Lösungen für alle UNIX- und Linux Derivate lauffähig sind. Die Portabilität der Lösungen hat einen hohen Stellenwert. Grundsätzlich kann unix4APLers als shell Erweiterung verwendet werden, indem die Software mit [pfadname/]unix4APLers.sh.inc in eine bestehende shell oder ein shell script eingebunden wird. Ideen und Lösungsansätze, welche in array programming languages wie APL, J oder K implementiert sind, sollen anderen Systemumgebungen zur Verfügung gestellt werden.

unix4APLers.sh.inc ist in erster Linie als Erweiterung des Funktionsumfanges für shell scripte entwickelt. Die Software kann gleichwohl als shell Erweiterung verwendet werden, wenn sie entweder manuell oder in einem .profile script ‚eingesourced' wird. Aus Gründen der Dokumentation und Hilfe ist die include Datei auch ausführbar wie ein Programm. Dann ist eine mehrstufige Dokumentation der Software unmittelbar möglich:

```
$ ./unix4APLers.sh.inc
# usage:unix4APLers.sh.inc                    Shows this help.
#       unix4APLers.sh.inc -?                 Shows overview for functions.
#       unix4APLers.sh.inc apl-function(s)    Shows help for the functions given as argument.
#       unix4APLers.sh.inc -??                Shows detailed help for all functions (huge output).

# Note: usually you may want to source in the unix4APLers.sh.inc utilities with

  . "/home/APLGERMANY/unix4APLers.sh.inc"
# to make the functions available to your script or application.

APLGERMANY@RNEEEPC ~
```

Man erhält also eine Liste der implementierten Funktionen mit: unix4APLers.sh.inc -?`

Und das Ergebnis ist derzeit wie folgt:

```
# List of functions:

  apl_general_purpose           ### Useful functions for nealy all purposes. ###
  apl_scriptname                    # Name of the script being executed.
  apl_dirname                       # Full path of scripts directory
  apl_getargv                       # get argc/argv from script or function (requires ksh)
  apl_getargv_eval                  # get argc/argv from script or function (bash and ksh)
  apl_show_argv                     # Display argc/argv set by apl_getargv
  apl_get_argv                      # Extract from argv set by apl_getargv (apl_io)
  apl_toupper                       # String conversion: translate to upper case.
  apl_tolower                       # String conversion: translate to lower case.
  apl_var_len                       # Return length of (string-)argument(s).
  apl_wordlist_len                  # Return length of wordlist.
  apl_io                            # Query or set Index Origin.
  apl_is_numeric                    # Is integer RC=0, otherwise RC>0
  apl_get_blanks                    # Get number of blanks requested in arg1
  apl_get_nulls                     # Get number of zeroes requested in arg1
  apl_qts                           # Query time stamp as YYYY MM DD HH MM SS (w/o milliseconds)
  apl_alarming_functions        ### Time controlled execution of commands. ###
  apl_set_alarm                     # Start an alarm: set_alarm n_seconds.
  apl_set_alarm_loop                # Start an alarm: set_alarm every n_seconds
  apl_reset_alarm                   # Reset an alarm (Terminate background task).
  apl_execute_with_timeout          # Execute a command unter timeout condition.
  apl_default_trap_handler          # Default trap handler for the apl_alarming_functions.
  apl_strings_and_arrays        ### Functions to handle strings and arrays. ###
  apl_sub_string                    # Substring from arg1, startpos $2 and length $3
  apl_sub_list                      # Words/numbers fromt arg1, startpos Arg2, length Arg3
  apl_extract_delimited_list        # Extract from Arg1 data between delimiters in Arg2
  apl_find_string                   # Find in arg1 position of arg2 (first match apl_io)
  apl_find_word                     # Find in alpha-/numeric vector arg1 position of arg2
  apl_find_last_in_string           # Find in arg1 position of arg2 (last match) (apl_io)
  apl_wordlist_sort                 # Sort elements in wordlist.
  apl_wordlist_unique               # Sort elements in wordlist and remove duplicates.
  apl_arithmetic                ### Functions for numeric problems. ###
  apl_abs                           # Return absolute value of an integer
  apl_add                           # Add arguments given (APL plusreduction)
  apl_plusred   { apl_add $*; }     # APL name for apl_add
  apl_subtract                      # Perform arg1 - arg2 - arg3 ...
  apl_mult                          # Multiply arguments given ( APL multiply reduction)
  apl_mod                           # Calculate reminder of two arguments given
  apl_faculty                       # Calculate faculty of argument
  apl_APLish_functions          ### Functions for APL like working. ###
  apl_ravel                         # Concatenate Array1 Array2
  apl_rho                           # Query length of an array; Reshape to array.
  apl_pick                          # Extract data from beginning or end.
  apl_drop                          # Drop array elements from left or right
  apl_iota                          # Get index vector or query index vector (apl_io).
  apl_random                        # APL ⍰?⍰ Get random numbers w/o replication (apl_io).
  apl_date_functions            ### Functions for calendar problems. ###
  apl_tojul                         # Calculate Julian day number from yyyy mm dd
  apl_tojulsec                      # Calculate Julian day number from yyyy mm dd in
                                    # seconds from universe
  apl_togreg                        # Calculate yyyy mm dd from Julian day number
  apl_julsectogreg                  # Calculate yyyy mm dd from Julian day number
  apl_datecalc                      # Calculate yyyy mm dd from Julian day number
  apl_dateinfo                      # Display information about a date given

APLGERMANY@RNEEEPC ~
```

In diesem Beitrag werden einige dieser Funktionen beispielhaft erläutert.

Numerische arrays und arrays aus Worten

Für das Indizieren von Vektoren haben sich zwei Methoden herausgebildet. In der Mathematik werden arrays im allgemeinen mit 1 beginnend indiziert. Im Bereich der Informatik hingegen ist es üblich, arrays mit einem Index von 0 beginnend zu indizieren, insbesondere dann, wenn der Anfang des arrays über einen Pointer unmittelbar erreichbar ist. Es ist also folgerichtig, beide Index Methoden zu unterstützen.

In APL ist dafür der Index Origin [] IO vorgesehen. Damit kann der Ursprung eines arrays, das erste array Element auf 0 oder 1 gesetzt werden. Die Implementierung im Rahmen des Projektes unix4APLers ist nachstehend angegeben.

Hinweis: Der Quellcode jeder Funktion kann mit dem Aufruf unix4APLers.sh.inc {funktionsname} ausgegeben werden.

```
# Description for function [apl_io]:

#<apl_io>                        # Query or set Index Origin.
# Usage:        apl_io | apl_io { 0 | 1 }
# Arguments:    Query index origin, if no argument is given
#                                   or set index origin to either 0 or 1.
# Example:      apl_io 1         # Arrays will be indexed starting with 1 as first element.
#               apl_io 0         # Arrays will be indexed starting with 0 as first element.
#               apl_io           # Querys index origin; result is ⎕IO  in this example.
#               my_io=`apl_io`   # Stores current index origin in variable  my_io .

# Source code for function [apl_io]:

apl_io(){
   [ "0" -eq "$#" ] && { [ -z "$APL_Qio" ] && APL_Qio="1"; echo "$APL_Qio"; return 0; }
   [ "1" -ne "$#" ] && echo "apl_io: Too many Arguments." && return 1
   [ "1" -eq "$1" ] && APL_Qio=1 && return 0
   [ "0" -eq "$1" ] && APL_Qio=0 && return 0
   echo "apl_io: Argument error[$1]." && return 1
}                                # Query or set Index Origin.
```

Beispiel der Arbeitsweise

Die Funktionen von unix4APLers.sh.inc werden in Linux oder unix durch einsourcen bereitgestellt. Die Methode ist:

```
. ./unix4APLers.sh.inc
```

Anschließend sind die Funktionen verfügbar.

```
$ apl_io
1

$ numvect1="1 2 3 4 5"

$ apl_sub_list "$numvect1" 2 2
2 3

$ apl_io 0

$ apl_sub_list „$numvect1" 2 2
3 4
```

Auf dem Gebiet der Formatierung von Ausgaben oder dem Parsen von Benutzereingaben wird viel Aufwand getrieben. Es werden unzählige Zeichen gezählt, welche abgeschnitten oder entfernt bzw. als Leerzeichen am Anfang oder am Ende der Zeichenkette angefügt werden sollen. An dieser Stelle könne zwei APL Primitives wertvolle Hilfe leisten. Diese sind: pick ‘ ↑‘ und drop ‘ ↓‘.

```
$ # Output formatieren

$ echo "`apl_pick 20 \"Eingabe1:
\"`Input1"
Eingabe1:           Input1

$ echo "`apl_pick 20 \"Zweite Eingabe:
\"`Input2"
Zweite Eingabe:     Input2

$ echo "`apl_pick -20 \"Eingabe1:
\"`Input1"
          Eingabe1: Input1

$ echo "`apl_pick -20 \"Zweite Eingabe:
\"`Input2"
    Zweite Eingabe: Input2

$ apl_drop 7 "prompt: Input"
 Input

$ apl_drop -6 "prompt: Input"
prompt:
```

Eine wesentliche Bedeutung bei dem Verarbeiten von arrays stellt das Interpretieren des Formates dar. So ist die Länge eines numerischen arrays durch die Anzahl der darin enthaltenen Zahlen bestimmt. Bei nicht numerischen arrays kann man die Länge des arrays interpretieren als die Anzahl der darin enthaltenen Zeichen oder als die Anzahl der darin enthaltenen ‚Worte‘. Mit dem APL primitive rho lassen sich die ersten beiden Fälle lösen. Das Interpretieren von nicht numerischen arrays als Wortliste ist gelöst analog dem Interpretieren von numerischen arrays. Nachfolgend wird ein Beispiel dazu gegeben:

```
# Numerischer array
numvect1="11 22 33 44 55"

apl_rho "$numvect1"
5

# Character array
charvect1="abc def ghi jkl"

apl_rho "$charvect1"
15

apl_wordlist_len "$charvect1"
4
```

Numerik

Exkurs: Natürlich ist auch das APL primitive ‚?‘, das erzeugen von Zufallszahlen, nach UNIX portierbar. Damit ist es leicht möglich, Verarbeitungsaufträge auf mehrere Server gleichmäßig zu verteilen und so ein einfaches load balancing zu realisieren. Die entsprechende Funktion apl_random kann etwa eine Zufallszahl oder mehrere Zufallszahlen ohne Wiederholung aus einem vorgegebenen Wertebereich ziehen. Ein typisches Samstagabend Beispiel ist etwa:

```
apl_io
1

apl_random 6 49
5 23 29 36 37 39
```

Aus dem Gebiet der Numerik soll an dieser Stelle das Problem der Kalender- und Zeitrechnung herangezogen werden. In Unix Betriebssystemen beginnt allgemein die Zeitrechnung mit dem 1. Januar 1970 0:00 Uhr UTC. Unter Windows und Windows Office gelten andere, ggf. release abhängige Regeln. Historisch bedingt ist die interne Darstellung der Datums- und Zeitinformation in einer Speicherplatz sparenden Form gegeben. So sind in etwa in perl derzeit nur Datums- und Zeitinformationen zwischen 1.1.1970 bis etwa zum Jahr 2100 möglich. Für eine generelle Zeitrechnung ist dies unbefriedigend.

In der Astronomie hat sich als eine universelle Systematik der Datum- und Zeitrechnung die julianische Tagesnummer durchgesetzt. Das Prinzip dabei ist die Darstellung des Kalenderdatums als eine ganze Zahl und der Uhrzeit als Bruchteil der Sekunden innerhalb des Tages und der Anzahl der Sekunden pro Tag (86400). Mit dem julianischen Kalender ist es möglich, beliebige Datums- und Zeitinformationen in jeden bekannten Kalender zu konvertieren.

Bei der Suche nach universal einsetzbaren Lösungen unter Unix waren die Ergebnisse nicht erfreulich. Für APL wurde rasch eine kompakte und funktionierende Implementierung von Peter Antweiler gefunden. Weiterführende Informationen für Kalenderberechnungen und das Konvertieren zwischen Kalendern von Peter Antweiler sind im APL-Journal von APL-Germany zu finden.

Mit Erlaubnis des Autors wurde die APL Lösung für eine Nutzung unter Unix-/ Linux- shells und der Windows DOS-box nach perl migriert. Für Leser, welche APL nicht kennen, ist nachstehend der APL code und der perl code angegeben. Die APL spezifischen Zeichen sind vielleicht nicht so gebräuchlich. Für nicht APLer soll gesagt werden, dass APL eigentlich entwickelt wurde, um mathematische Algorithmen kompakt zu formulieren. Die Implementierung als eigene Entwicklungs- und Anwendungsumgebung, etwa als Dyalog APL oder IBM APL, stellt letztendlich ein System zur Ausführung von in APL Syntax beschriebenen Algorithmen dar.

Der APL code:

```
      ∇JDFROMDATE[⎕]∇
[0]   JD←CAL JDFROMDATE DATE;UT;D;BIT;A;B;BR
[1]   ⍝ COMPUTE JULIAN DAY NUMBER FROM DATE
[2]   ⍝ CAL = 'J':  DATE IS IN THE JULIAN
CALENDAR
[3]   ⍝ CAL = 'G':  DATE IS IN THE GREGORIAN
       CALENDAR
[4]   ⍝ CAL = 'A':  THE GREGORIAN CALENDAR
REFORM IS
[5]   ⍝ TAKEN INTO ACCOUNT: THE DAY FOLLOWING
[6]   ⍝ 1582 OCT 4 IS 1582 OCT 15.
[7]   ⍝ DATE IS VECTOR OR MATRIX, COLUMNS ARE:
[8]   ⍝ YEAR, MONTH, DAY, HOUR, MINUTE, SECOND
[9]    JD←⍳0
[10]   ⍎(2≠⎕NC ,CAL')/'CAL←''A'''
[11]   ,→0' ⎕EA , AI←÷∨/''AJG''=CAL←↑CAL'
[12]   DATE←6 MATRIX DATE
[13]   (0 1↓DATE)←|0 1↓DATE
[14]   DATE[;3]←DATE[;3]-BR←1|DATE[;3]
[15]   D←10000 100 100⊥⍉DATE[;⍳3]
[16]   DATE[;1]←DATE[;1]-BIT←DATE[;2]≤2
[17]   DATE[;2]←DATE[;2]+BIT×12
[18]   UT←BR++/DATE[;4 5 6]÷[2]24 1440 86400
[19]   B←2+(⌊A÷4)-A←⌊DATE[;1]÷100
[20]   B←0,[1.5]B ⍝ B[;1]=JUL. B[;2]=GREG.
```

```
[21]   →(CAL='A')/L10
[22]   B←,B[;('JG'=CAL) /1 2]
[23]   →L20
[24]  L10:B[;2]←B[;2]×D≥15821015
[25]   B←+/B
[26]  L20:JD←¯1524.5+DATE[;3]+B+⌊365.25×DA
        TE[;1]+4716
[27]   JD←JD+UT+⌊30.6001×1+DATE[;2]
       ∇ 2004-02-15 05.13.33 (GMT+1)
```

Und die Lösung in perl:

```
#<apl_tojul>                                  # Calculate Julian day number from yyyy mm dd
# Usage:        apl_tojul Dateinfo            # Given as yyyy mm dd
# Arguments:    Dateinfo
# Example:      apl_tojul 2008 3 19 22 48 0   # Result „2454545.45"

# Source code for function [apl_tojul]:

apl_tojul(){ # rnus: perl version of Peter Antweilers „JDFROMDATE"
   [ „3" -gt „$#" ] && return 1
   apl_args=$@
   apl_args=`apl_pick 6 "$apl_args"`
   set -- $apl_args
   perl -e '
      #!/usr/bin/perl
      # apl_tojul.pl yyyy mm dd [ HH MM SS ] -> julian day number
      $y = 0 + @ARGV[0];
      $m = 0 + @ARGV[1];
      $d = 0 + @ARGV[2];
      $D = 10000*$y + 100*$m + $d;
      $hh= 0 + @ARGV[3];
      $mm= 0 + @ARGV[4];
      $ss= 0 + @ARGV[5];
      if ( $m <= 2 ) { $BIT = 1; } else { $BIT = 0; }
      $y = $y - $BIT;
      $m = $m + $BIT * 12;
      $UT = ( $hh*3600 + $mm*60 +$ss ) / 86400;
      if ( $D >= 15821015 ) { $A = int( $y / 100 ); $B = 2 + int( $A / 4 ) - $A; } else { $B = 0; }
      $JD = $d+$B+int(365.25*($y + 4716))-1524;
      $JD = $JD + int( 30.6001 * $m ) + $UT - 0.5;
      $JD = $JD + 31; # ??
      if ( 59  > $JD ) { $JD=$JD+1; }  # -4712 was no leap year
      if ( 5 == $m || 7 == $m || 10 == $m || 12 == $m ) { $JD = $JD - 1; } # bug fix
      print "".$JD."\n";
   , "$1" "$2" "$3" "0$4" "0$5" "0$6"
}                                                 # Julian day number from Date/Time
```

Mit der Funktion apl_tojul wird ein Datum-/Uhrzeit Wert in die julianische Darstellung umgewandelt. Das Gegenstück dazu ist die Funktion apl_togreg, welche einen Wert von der julianischen Darstellung in die Form des bei uns gebräuchlichen gregorianischen Kaldender umwandelt. Aus Platzgründen wird hier auf die Angabe des Quellcodes verzichtet.

Zur Praxis, dazu einige Beispiele:

```
$ apl_tojul 2008 11 25
2454795.5

$ apl_tojul 2008 11 25 12 00 00
2454796

$ apl_togreg 2454795.5
2008 11 25 0 0 0

$ apl_togreg 2454796
2008 11 25 12 0 0
```

```
# Interessantes Beispiel
$ now="2008 11 12 20 39 55"

$ apl_tojul $now
2454783.36105324

$ apl_togreg 2454783.36105324
2008 11 12 20 39 54
```

Wenn wir eine beliebige Uhrzeit als Bruch darstellen wollen, sind Rundungsfehler nicht vermeidbar,weil bei der Division durch 86400, der Anzahl Sekunden je Tag, eine 3 im Nenner ist. Es ist mit dem julianischen Kalender also nicht möglich, eine Datum-/Uhrzeit Information ohne Verlust an Information darzustellen.

Für eine Darstellung von Datum- und Uhrzeitinformationen wird - zumindest für die interne Darstellung - folgende Methode vorgeschlagen: Moderne Computer sind in der Lage, mit großen Integer Werten umzugehen. Was hindert uns daran, generell in Sekunden zu rechnen. Wenn die Tagesnummer des julianischen Datums mit 86400 (Sekunden pro Tag) multipliziert wird, können die Sekunden innerhalb eines Tages unmittelbar addiert werden. Dann passieren keine Rundungsfehler und es entsteht kein Verlust an Information.

Das modifizierte perl Programm sieht dann wie folgt aus:

```
#<apl_tojulsec>                                  # Calculate Julian day number from yyyy mm dd in
seconds from universe
# Usage:        apl_tojul Dateinfo               # Given as yyyy mm dd
# Arguments:    Dateinfo
# Example:      apl_tojul 2008 3 19 22 48 0      # Result "2454545.45"

# Source code for function [apl_tojulsec]:

apl_tojulsec(){ # rnus: perl version of Peter Antweilers „JDFROMDATE"
   [ "3" -gt "$#" ] && return 1
   apl_args=$@
#   apl_args=`apl_pick 6 "$apl_args"`
   set -- $apl_args
   perl -e '
      #!/usr/bin/perl
      # apl_tojul.pl yyyy mm dd [ HH MM SS ] -> julian day number
      $y = 0 + @ARGV[0];
      $m = 0 + @ARGV[1];
      $d = 0 + @ARGV[2];
      $D = 10000*$y + 100*$m + $d;
      $hh= 0 + @ARGV[3];
      $mm= 0 + @ARGV[4];
      $ss= 0 + @ARGV[5];
      if ( $m <= 2 ) { $BIT = 1; } else { $BIT = 0; }
      $y = $y - $BIT;
      $m = $m + $BIT * 12;
#      $UT = ( $hh*3600 + $mm*60 +$ss ) / 86400;
      $UT = ( $hh*3600 + $mm*60 +$ss );
      if ( $D >= 15821015 ) { $A = int( $y / 100 ); $B = 2 + int( $A / 4 ) - $A; } else { $B = 0; }
      $JD = $d+$B+int(365.25*($y + 4716))-1524;
#      $JD = $JD + int( 30.6001 * $m ) + $UT - 0.5;
      $JD = $JD + int( 30.6001 * $m ) - 0.5;
      $JD = $JD + 31; # ??
      if ( 59  > $JD ) { $JD=$JD+1; }  # -4712 was no leap year
      if ( 5 == $m || 7 == $m || 10 == $m || 12 == $m ) { $JD = $JD - 1; } # bug fix
      $JD = ($JD*86400) + $UT; # inserted to calcluate in seconds
      print "".$JD."\n";
   ' "$1" "$2" "$3" "0$4" "0$5" "0$6"
}                                                  # Julian day number from Date/Time
```

Ein APLer flunkert nicht. Anbei der Beweis, dass obiges Problem des Rundungsfehlers behoben ist:

```
$ $ECHO "$now"
2008 11 12 20 39 55

$ apl_tojulsec $now
212093282395

$ apl_julsectogreg 212093282395
2008 11 12 20 39 55
```

Mit diesen Werkzeugen ist es nun möglich, über lange Zeiträume Datum- und Uhrzeit Differenzen zu ermitteln. Es wurde eine Funktion apl_datecalc implementiert, mit folgendem Funktionsumfang (Der source code kann wie oben angegeben, angezeigt werden):

```
#<apl_datecalc>                                  # Calculate date differences
# Usage:        apl_datecalc yyyymmddhhmmss {+|-|dateminus} {num|num|yyyymmddhhmmss}\
                    {week[s]|day[s]|hour[s]|minute[s]|second[s]}
# Arguments:    yyyymmddhhmmss          # human readable time stamp
#               {+|-|dateminus}         # Option1 +: add to time stamp, \
                                              -: subtract, \
                                                dateminus:  calculate time differences
#               num                     # unsigned integer
#               {week[s]|day[s]|hour[s]|minute[s]|second[s]} # Unit in which time\
                                        difference should be presented.

# Beispiele dazu:

$ apl_datecalc 20081125 dateminus 19520905 days
apl_tojulsec 1952 09 05 00 00 00 jd2=210320107200
apl_tojulsec 2008 11 25 00 00 00 jd=212094331200
20535

$ apl_datecalc 20081125 dateminus 19520905 hours
apl_tojulsec 1952 09 05 00 00 00 jd2=210320107200
apl_tojulsec 2008 11 25 00 00 00 jd=212094331200
492840

$ apl_datecalc 20081125 dateminus 19520905 minutes
apl_tojulsec 1952 09 05 00 00 00 jd2=210320107200
apl_tojulsec 2008 11 25 00 00 00 jd=212094331200
29570400

$ apl_datecalc 20081125 dateminus 19520905 seconds
apl_tojulsec 1952 09 05 00 00 00 jd2=210320107200
apl_tojulsec 2008 11 25 00 00 00 jd=212094331200
1774224000

$ apl_datecalc 20081125 - 1 seconds
20081124235959

$ apl_datecalc 20081125 - 1 minute
20081124235900

$ apl_datecalc 20081125 - 1 week
20081118000000

$ apl_datecalc 20081125 - 180 days
20080529000000
```

Es mag sein, dass die verlustfreie Berechnung von Datum-/Uhrzeit-informationen in langen Zeiträumen eher von astronomischer oder akademischer Bedeutung ist.

Aber, wenn die letzten vier Beispiele betrachtet werden, dann sieht man sofort, dass bei kurzen Zeitintervallen etwa dem Problemen von alerts zu gleichen Problemen entgegengewirkt werden kann. Wenn beispielsweise ein Server auf einen Fehler läuft, dann sind als Konsequenz davon auch alle seine Datenbanken nicht mehr verfügbar. Anstelle von ‚vielen' Alarmmeldungen, welche gegebenenfalls ‚viel' Personal binden, ist es hinreichend, zu melden, ‚Ein Rechner hat ein Problem', als ‚Rechner1 hat ein Problem, Datenbank1 hat ein Problem, Datenbank2 hat ein Problem, Datenbank3 hat ein Problem, ...'. Die gleiche Problematik stellt sich im Internet bei denial of service attacs.

Eine Berechnung für größere Zeiträume kommt insbesondere in Bezug auf das Löschen nicht mehr benötigter Dateien (housekeeping) in Frage. Auch wenn Plattenplatz mittlerweile in Terabyte gerechnet wird, der Speicherplatz ist endlich. Es ist und wird unumgänglich sein, alte Dateien zu löschen. Wenn man also das Alter einer Datei bezogen auf ‚jetzt' kennt, so kann einfach entschieden werden, ob diese als obsolet betrachtet werden soll oder nicht.

Verarbeiten von Events

Damit Anwendungen stabil laufen, müssen diese auch mit unerwarteten Situationen umgehen können. Als ein Beispiel dazu wird die zeitliche Überwachung von Anwendungen vorgestellt.

Typische Aufgabenstellungen sind etwa Netzanwahl Prozeduren. Hier wird beispielsweise als Vorgabe ein Wert von 10 Sekunden gesetzt, bis die Anwahl erfolgt sein soll. Falls dies nicht der Fall ist, wird von einem Netzwerkfehler ausgegangen und die weitere Verarbeitung abgebrochen.

Ein anderes Anwendungsgebiet ist die Überwachung von lange laufenden Prozessen. Hier kann es sinnvoll sein, etwa nach 2 Stunden eine Warnung auszugeben, damit eventuelle Probleme untersucht werden können. Als Aktionn kommt eventuell eine Veränderung von Job Prioritäten in Betracht, um die gesamte Verarbeitung erfolgreich zu Ende zu bringen.

Methodisch wird dabei so vorgegangen, dass eine Überwachungstask gestartet wird, welche nach einem als Parameter mitgegebenen Zeitabschnitt, einen event an die rufende Task sendet. In Dyalog APL wird für das Senden von events der Befehl []signal verwendet. Unter Unix wird der Befehl kill, gefolgt von einer Signal Nummer benutzt.

Die rufende Task ist mit einem Programmabschnitt, dem trap handler, ausgestattet, welche diese events verarbeiten können. In Dyalog APL werden solche Programmteile mit dem []trap Befehl definiert. Unter Unix heißt der Befehl ebenfalls trap.

Innerhalb des trap handlers kann eine beliebige Verarbeitung stattfinden. An dieser Stelle ist es möglich, Warnungen auszugeben, etwa TEC Meldungen, oder Mails bzw. sms Meldungen zu versenden. Darüber hinaus ist es möglich, die gerade laufende Task zu beenden; dies wäre die Aufgabe bei dem ersten Beispiel von oben.

Eine generelle Funktion benötigt also folgende Informationen, welche als Parameter bereitgestellt werden:

- Den oder die Befehle, welche auszuführen sind
- Das zu beachtende Zeitlimit
- Den Typ von Alarm (einmaliger oder wiederholter Alarm)
- Den trap handler (Befehle, welche im Alarmfall ausgeführt werden)

Es ist möglich, eine sehr allgemeine und kompakte Funktion zu erstellen, welche diese Probleme wie folgt löst. Prinzipiell sind nur vier Schritte erforderlich, um die alarming Aufgabe zu lösen:

- Der trap handler ist zu aktivieren
- Die alarming subtask ist entsprechend dem mitgegebenen Parameter als einmaliger oder wiederholter Alarm zu starten
- Das oder die zu überwachenden Kommandos, welche als Parameter gegeben wurden, sind auszuführen
- Die alarming subtask ist zu beenden, falls diese noch aktiv ist

Hier ist das Beispiel für die Implementierung:

```
# - command(s) given as argument1
# - with a time limit in seconds given as argument 2
# - alarmtype "" or "loop" given as argument 3
# - using a trap handler given as argument 4.
# Command(s) continue processing or will be terminated,
#            depending on the trap handler.
apl_execute_with_timeout(){
   [ "4" -ne "$#" ] && { echo "execute_with_timeout - invalid arguments."; exit 1; }

   # Enable trap for sigint for postprocessing if necessary.
   eval trap "'$4'" $APL_ALRM

   [ "loop" = "$3" ] && apl_set_alarm_loop "$2" >/dev/null 2>&1 || \
                        apl_set_alarm "$2" >/dev/null 2>&1 # Set alarm - $2 seconds.

   eval "$1"       # Execute commands from $1; Expect finishing within $2 seconds.

   apl_reset_alarm # Reset alarm.

}
```

Das Bearbeiten von traps als Ausnahmesituationen ist natürlich höchst anwendungsspezifisch. Es ist dennoch möglich, eine generelle Vorgehensweise zu beschreiben. An Reaktionen auf eine Ausnahmesituation kommt in Betracht,

- Protokollieren des events als Meldung auf stdout oder in einer log Datei
- Setzen einer Variablen in der rufenden Funktion, damit ggf. eine alternative Verarbeitung stattfinden kann
- Veranlassen von Alarmaktionen wie das Erstellen von TEC Meldungen oder das Versenden emails oder sms
- für wiederholte events können eskalierende Alarmaktionen implementiert werden
- Beenden der bzw., der überwachten Kommandos, falls dies so vorgegeben wurde

Damit ergibt sich für die Implementierung als erster Ansatz:

```
apl_trap_handler_ALRM=`
   echo "SIGALRM received." >&2          # Give information to stderr.
   APL_ALARM_TIMEOUT="yes"                # Inform calling function/main script.
                                          # optionally issue other events (TEC).
                                          # Do postprocessing/cleanup before exit.
   return                                 # Terminate current process/function.
`

apl_trap_handler_ALRM_WARNING=`
   echo "SIGALRM received. Timeout." >&2 # Give information to stderr.
   APL_ALARM_TIMEOUT="yes"                # Inform calling function/main script.
                                          # optionally issue other events (TEC).
`                                         # Processing continues.

apl_trap_handler_ALRM_WARNING_LOOP=`
   echo "SIGALRM received. Timeout." >&2 # Give information to stderr.
   APL_ALARM_TIMEOUT="yes"                # Inform calling function/main script.
                                          # optionally issue other events (TEC).
   let APL_ALARM_COUNT=$APL_ALARM_COUNT+1

   case $APL_ALARM_COUNT in
      1)
         echo "1st Alarm."
      ;;
      2)
         echo "2nd Alarm."
      ;;
      3)
         echo "3rd Alarm."
      ;;
      *)
         echo "Plenty of Alarms."
      ;;
   esac
`                                         # Processing continues.
```

Für die eigentliche Zeitüberwachung und das Absetzen des/der alarm events kann als einfache Methode das Starten einer subtask implementiert werden. Im einfachsten Fall umfasst die subtask eine Pause mit der gewünschten Zeitdauer, gefolgt von dem Auslösen des events. Bei der Implementierung von wiederholten events ist das Ganze nur in eine Endlosschleife zu verpacken. Falls die zu überwachende Anwendung schnell genug abgearbeitet ist, muss nur noch die alarm subtask beendet werden. Die dazugehörende task nummer wird zweckmäßigerweise bei deren Start zwischengespeichert.

Die Implementierung sieht folgendermassen aus:

```
#<apl_set_alarm>                # Start an alarm: set_alarm n_seconds.
# Usage:    apl_set_alarm
# Arguments:    Number              # Number of seconds, when alarm is issued.
# Example:  apl_scriptname 10       # Issue an alarm after 10 seconds.
#</apl_set_alarm>
apl_set_alarm(){                    # Start an alarm: set_alarm n_seconds
   sh -c "sleep "$1"                # Sleep $1 seconds.
          kill -s $APL_ALRM "\$PPID"     # Send SIGALRM to parent.
          exit 0"  >/dev/null 2>&1 & # Backgrounding alarm.

   if [ 0 -ne "$?" ]; then
      echo "Cannot start alarm."
      exit 1
   fi
   APL_ALARM_PID="$!"               # Remember background's pid.
   APL_ALARM_TIMEOUT="no"           # Thinking positive.
}

#<apl_set_alarm_loop>               # Start an alarm: set_alarm every n_seconds
# Usage:    apl_set_alarm_loop
# Arguments:    Number              # Number of seconds, when alarm is issued.
# Arguments:    Number              # Maximum loop count.
# Example:  apl_scriptname 10       # Issue an alarm after every 10 seconds.
#</apl_set_alarm_loop>
apl_set_alarm_loop(){               # Start an alarm: set_alarm every n_seconds
   sh -c „while [ `1` -eq `1` ]; do
             sleep "$1"             # Sleep $1 seconds.
             kill -s $APL_ALRM "\$PPID"     # Send SIGALRM to parent.
           done
           exit 0" >/dev/null 2>&1 & # Backgrounding alarm.

   if [ 0 -ne "$?" ]; then
      echo "Cannot start alarm."
      exit 1
   fi
   APL_ALARM_PID="$!"               # Remember background's pid.
   APL_ALARM_TIMEOUT="no"           # Thinking positive.
   APL_ALARM_COUNT="0"
}

#<apl_reset_alarm>              # Reset an alarm (Terminate background task).
# Usage:    apl_reset_alarm
# Arguments:    -none-
# Example:  apl_reset_alarm         # Terminate a running alarm subtask.
#</apl_reset_alarm>
apl_reset_alarm(){                  # Reset an alarm (Terminate background task).
   [ -z "$APL_ALARM_PID" ] && return 0 # Nothing to do.
   ps |grep "^ *$APL_ALARM_PID" >/dev/null 2>&1 && kill -s KILL "$APL_ALARM_PID" >/dev/null 2>&1
   APL_ALARM_PID=
}
```

Zusammenfassung

Es wurde gezeigt, dass array- orientierte Methoden vielfältige Anwendungsgebiete auch in einfachen shell Umgebungen haben. Gerade bei dem Umgang mit Zeichenketten ergeben sich, wie gezeigt wurde, sehr nützliche Lösungswege. Auch bei dem Interpretieren von Argumenten auf der Kommandozeile ergeben sich Vorteile.

Auf dem Gebiet der Arithmetik lassen sich einfache, bewährte APL Funktionen leicht auf eine Unix Umgebung migrieren. Bei komplexen Lösungen wird eine Unix shell jedoch überfordert sein. Arithmetik mit großen numerischen Arrays sollte besser in einer ‚richtigen' Programmiersprache implementiert werden. Im systemnahen Bereich wurde auf dem Gebiet des Behandelns von Ausnahmesituationen (signal und trap) eine sehr einfache und flexible Lösung dargestellt, um beliebige Anwendungen zeitgesteuert zu überwachen. Das zeitliche Überwachen von Anwendungen, etwa das Erkennen von ‚Langläufern', verbunden mit Alarmierungsmechanismen ist in nahezu jedem Rechenzentrum zu behandeln. Dies ist besonders wichtig, wenn Abhängigkeiten zwischen den einzelnen Abläufen bestehen.

Insgesamt gesehen, wird die Problemlösung dadurch erleichtert, dass die hier zur Verfügung gestellten Funktionen die Schreibarbeit reduzieren. Da die Funktionen in dem Paket getestet wurden, verringert sich auch die Fehleranfälligkeit von neuen Anwendungen, wenn die vorhanden und getesteten Funktionen wieder verwendet werden.

Kontakt zum Autor:

Reiner Nussbaum, Dr. Nußbaum gift mbH,
Buchenerstraße 78, D-68259 Mannheim

E-Mail: info@nussbaum-gift.de
Internet: www.nussbaum-gift.de/
Tel.: 0621 / 71 52 190

Rolf Herminghaus

Fehlersuche mit APL+WIN 9.1

Die allgemein steigende Komplexität von Software erzwingt mehr und mehr auch neue Testmethoden, um die Funktionsfähigkeit der erstellten Software zu überprüfen. Hand in Hand damit wächst der Druck, Methoden zur Eingrenzung und Untersuchung möglicher Fehler stärker als bisher zu automatisieren. Der vorliegende Artikel behandelt einige Features der Sprache APL und darauf basierende einfache Hilfsprogramme, die die Fehleranalyse in komplexen Software-Systemen deutlich erleichtern können. Basis für die folgenden Darstellung ist die Sprache APL+WIN, die von der Firma APL2000 (ehem.Manuguistics) angeboten wird. Für die Realisierung der beschriebenen Tools fand die Version 9.1 Verwendung. Den Hintergrund bilden Erfahrungen beim Test mathematischer Module aus dem Bereich der Lebens- und Rentenversicherung.

Vorgehensweise Massentest

Eine Möglichkeit, sich der Test-Problematik von komplexen Softwaresystemen zu nähern, ist das Konzept des Massentests. Dabei wird auf Basis einer irgendwie gearteten Vorgabe die gleiche Funktionalität von unterschiedlichen Programmier-Teams unabhängig voneinander zweimal umgesetzt. In der darauf folgenden Testphase wird dann eine möglichst große Anzahl von Testfällen von beiden Rechenkernen verarbeitet und die Ergebnisse maschinell gegeneinander abgeglichen. Dort, wo Differenzen auftreten, ist mindestens eine der beiden Implementierungen fehlerhaft.

Vorteile des Massentest-Verfahrens

- Reduktion von manuellen Untersuchungen auf die Fälle, in denen tatsächlich Differenzen aufgetreten sind
- „Natürliche" Möglichkeit, Regressionstests zu fahren
- Erhöhung der Testfall-Anzahl ohne signifikante Erhöhung des Testaufwandes

Nachteile des Massentest-Verfahrens

- Notwendigkeit der Doppelprogrammierung und damit erhöhter Aufwand

Die Doppelprogrammierung ist auch der Grund für die geringe Akzeptanz, die diese Testmethodik bei manchen altgedienten EDV-Profis immer noch findet. Man muss aber davon ausgehen, dass es im kommerziellen Bereich kaum andere Möglichkeiten gibt, hochkomplexe Programme mit realistischem Aufwand auf Korrektheit bzw. Fehler hin zu untersuchen.

Insbesondere im Bereich der Lebensversicherung mit ihren komplexen mathematischen Algorithmen scheint sich das Verfahren daher mehr und mehr durchzusetzen.

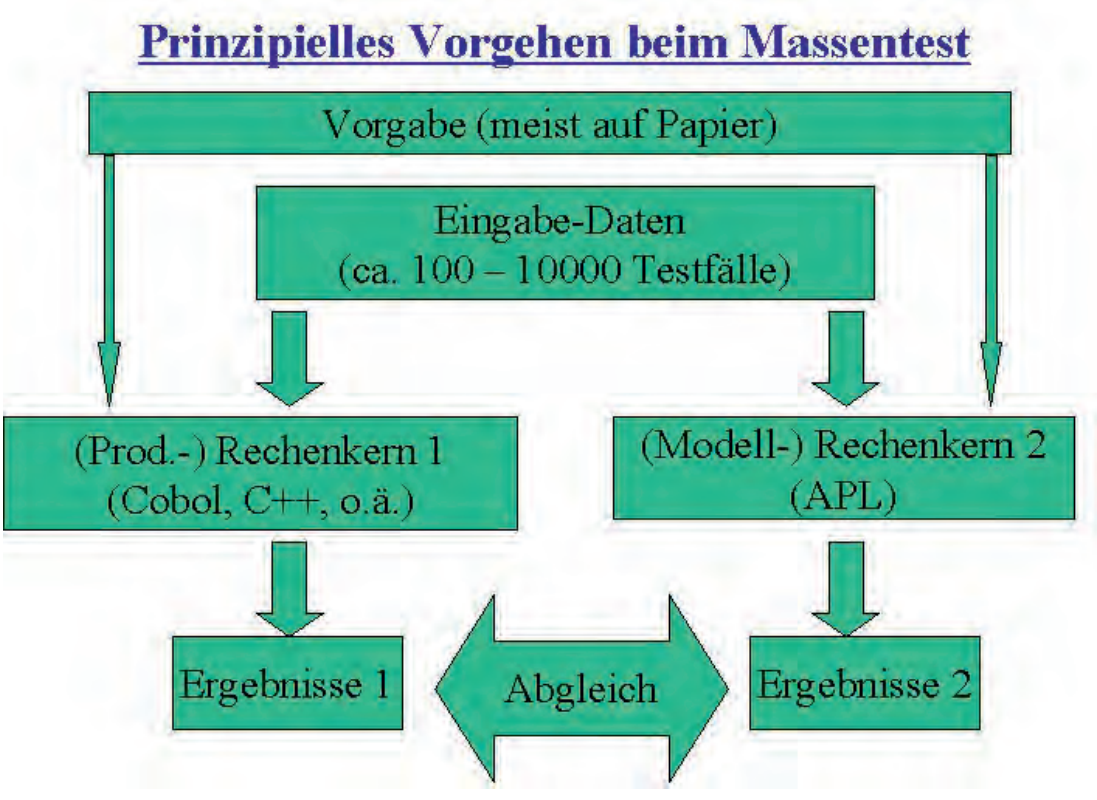


Abb. 1: Massentest

Warum APL als Implementationssprache für den Modell-Rechenkern ?

- APL-Programme lassen sich sehr schnell schreiben und extrem gut debuggen.

- komplexe mathematische Algorithmen lassen sich in APL einfacher als in anderen Programmiersprachen realisieren.

- Änderungen können leicht – sogar während das Programm läuft – durchgeführt werden.

- Wegen der Workspace-Eigenschaft der Sprache können Änderungen „versuchsweise" in einer Kopie des Rechenkerns durchgeführt werden, ohne bereits Erreichtes zu gefährden.

- Kollegen können eine Kopie des erreichten Standes für eigene

Untersuchungen nutzen, ohne von den aktuell laufenden Tests und Änderungen tangiert zu werden.

Motivation für die Entwicklung unterstützender Werkzeuge

- Der APL-Rechenkern ist groß und unübersichtlich (typischerweise 1500 Programme / 500 Variablen).
- Große Teile des APL-Rechenkerns stammen von anderen Kollegen – sind also für den Tester „unbekanntes Gelände".
- Anzahl und Komplexität der vorhandenen Fehler in „Rechenkern 1" und „Rechenkern 2" sind zu Beginn der Testphase völlig unbekannt und lassen sich auch nur unzureichend abschätzen.
- Der Termin für die Freigabe des produktiven Rechenkerns lässt sich in der Regel – oft aufgrund von gesetzlichem Zwang – nicht verschieben.

Unterstützende Features bei APL+WIN 9.1

Wie alle Programmiersprachen verfügt APL+WIN über einen Debugger – der in diesem Fall „Code-Walker" heißt. Daneben hat man die üblichen Möglichkeiten, STOP-Vektoren zu setzen, Step-by-Step-Durchläufe durchzuführen, TRACE-Ausgaben anzustoßen oder Variablennamen zu überwachen (Watchpoints).

Daneben bietet APL+WIN aber auch einige Features, die zur Eingrenzung von Fehlern genutzt werden können:

(⎕NL, ⎕MF, ⎕SI, ⎕CR, ⎕WATCHPOINTS, ⎕COPY)

1) Ausnutzung von ⎕MF für Debugging-Zwecke

Gedacht ist ⎕MF eigentlich als Mittel für Performancemessungen.

Vergleichbar ist diese Funktion etwa der Funktion ⎕PROFILE , die im Dyalog-APL zur Verfügung gestellt wird. Nach der Initialisierung misst ⎕MF für alle durchlaufenen Zeilen den Brutto- und Netto- Zeitaufwand, aber - und das ist in unserem Zusammenhang wesentlich – auch die Anzahl der Durchläufe.

Insbesondere ist es so möglich, mit Hilfe von ⎕MF die Zeilen im Workspace zu identifizieren, die während eines Tests überhaupt durchlaufen wurden.

Als Analyse-Ergebnis liefert ⎕MF zu jedem Funktionsnamen eine Nx3-Matrix mit den Zeitaufwänden „Brutto" (incl. Unterfunktionen) „Netto" (ohne Unterfunktionen) sowie die Anzahl der Durchläufe (N ist dabei zu verstehen als Anzahl der Funktionszeilen). Eine ganze Reihe einfacher Analyse-Funktionen lassen sich so realisieren.

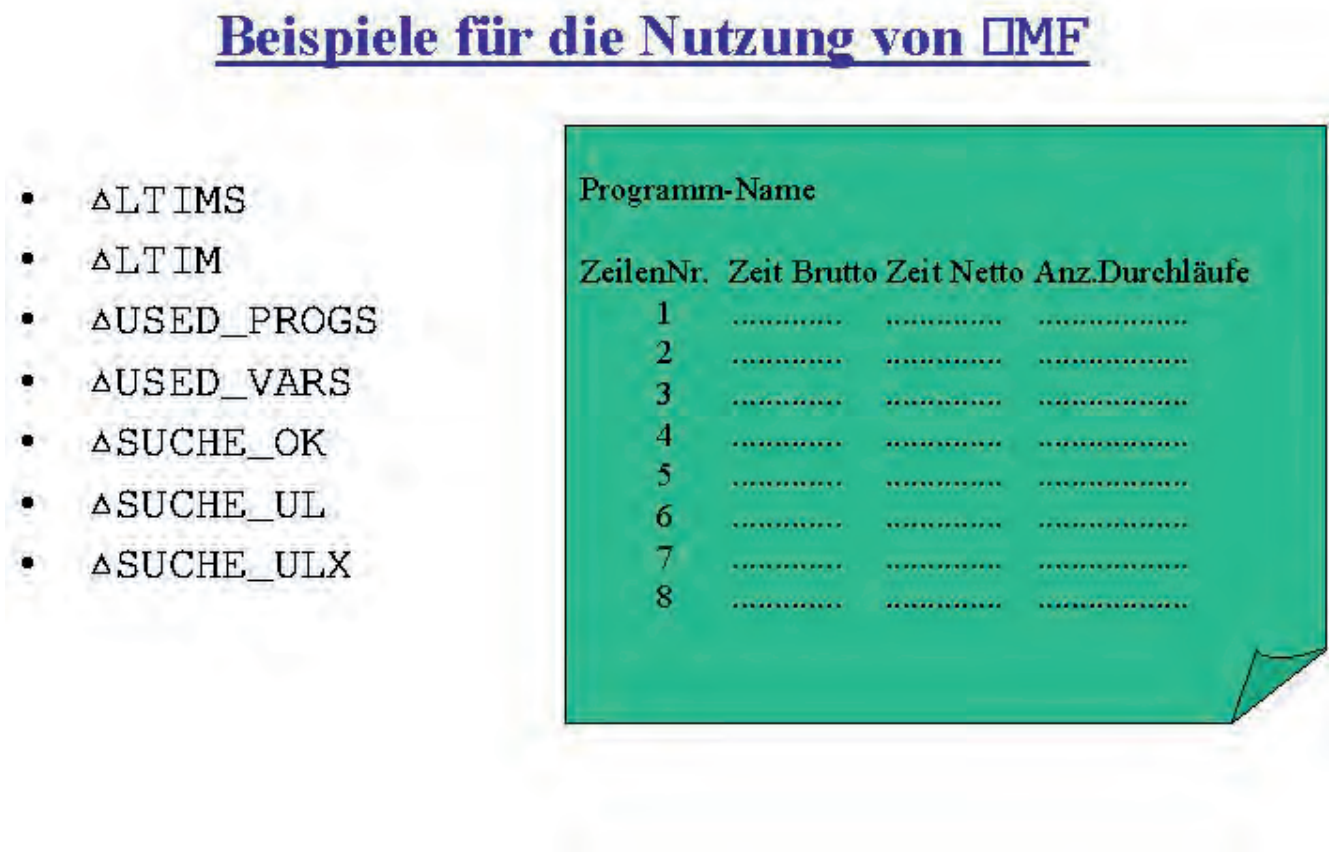


Abb. 2: Nutzung von ⎕MF

Beispiele:

```
⍝---Monitor-Funktion initialisieren----
     DUMMY←1 ⎕MF ⎕NL 3

⍝---Aufruf der zu testenden Funktion---
     a4←FORM_BB_PGM TEST_EING



⍝---Beispiel-Ergebnis ⎕MF -------------
     a5←⎕MF 'BB_B13_ABUZ'
           89977 4972 1
              77    7 1
              46   46 1
              12   12 1
               0    0 0
          ...etc.......
```

‘USED_PROGS und ‘USED_VARS

Bereits mit Hilfe dieser einfachen Funktionen läßt sich der Kreis der zu untersuchenden APL-Objekte oft schon entscheidend eingrenzen:

```
⍝---Liste der benutzten Unterprogramme-------
      a6← USED_PROGS

⍝-----------------------------------------
⍝ Liste der benutzten globalen Variablen
⍝ (Neben ⎕MF werden hier auch die
⍝ Systemfunktionen ⎕CR und ⎕NL ausgenutzt)
⍝-----------------------------------------
      a7←∆USED_VARS
```

‘LTIMS und ‘LTIM

Bei den Funktionen *∆LTIMS* und *∆LTIM* handelt es sich um Profiling-Funktionen, die die Daten von ⎕MF mit der ⎕CR-Darstellung der jeweils untersuchten Funktion zu einer Gesamt-Darstellung kombinieren.

```
⍝---Beispiel: ∆LTIMS ---
      a8←∆LTIMS ‘BB_B06_BARW‘
 BB_B06_BARW[65]  .              ⍝ ---- EINGABEWERTE FUELLEN ...
 BB_B06_BARW[66]  .              ⍝
 BB_B06_BARW[67]  2   26.03 26.03 (TETARIF TEKALKZ VX LX NX RX NXRZ ...
 BB_B06_BARW[68]  2   37.18 37.18 (TASU TAJAHR TEVSU TEBEG TEEAVP .....
 BB_B06_BARW[69]  2   32.25 32.25 (REENDAUF REENDGAR REZW RZEA .....
 BB_B06_BARW[70]  2   21.05 21.05 (FNKZ FNEA FNENDA FNREZW)←EING[26↓⍳30]
 BB_B06_BARW[71]  2   12.32 12.32 (CL8ENDA F_CODE TAKZ)←EING[31 32 33]
 BB_B06_BARW[72]  .              ⍝
 BB_B06_BARW[73]  2   14.22 14.22 TETARIF←7↑TETARIF,‘        ‘
 BB_B06_BARW[74]  2   10.40 10.40 TEKALKZ←6↑TEKALKZ,‘       ‘
 BB_B06_BARW[75]  2   33.42 33.42 VX←140↑VX,140⍴0
 BB_B06_BARW[76]  2   30.02 30.02 LX←140↑LX,140⍴0
 BB_B06_BARW[77]  2   27.50 27.50 NX←140↑NX,140⍴0
      ....etc..........
```

‘LTIM unterscheidet sich lediglich dadurch, daß hier die Angaben zum Zeitbedarf der einzelnen Zeilen weggelassen sind. ‘LTIM dient also nur dazu, einen schnellen Überblick über den Kontrollfluss innerhalb der untersuchten Funktion zu bekommen.

```
∆SUCHE_OK, ∆SUCHE_UL, ∆SUCHE_ULX
```

Die drei oben angegebenen SUCHE-Funktionen durchsuchen die Programm-Texte des Workspaces nach dem im Argument mitgegebenen String . Unterschiede hierbei:

```
⍝---Beispiel:  SUCHE_OK ---
     a12← SUCHE_OK ⊤TEENDAV
Funktion: BB_B04_PLAU
BB_B04_PLAU[247] :IF TEENDAV>ENDAMAX
BB_B04_PLAU[264] :IF TEEAVP_G≥TEENDAV

Funktion: BB_B06_BARW
BB_B06_BARW[ 68] (TASU TAJAHR TEVSU TEBEG TEENDAV PERDAT ....
BB_B06_BARW[102] TEENDAV←↑TEENDAV,0
BB_B06_BARW[156]     HV_LBW←BB_B06_AXN VX[2] TEENDAV NX NZNEU ....
BB_B06_BARW[343] HV_NBW←BB_B06_AEXN VX[2] NX NZNEU TEEAVP TEENDAV 1
.... etc. ....
```

∆SUCHE_OK
durchsucht ALLE Zeilen des Workspaces

∆SUCHE_UL
durchsucht lediglich die im Test durchlaufenen Zeilen des Workspaces

∆SUCHE_ULX
durchsucht lediglich die Zeichen links vom Zuweisungs-Pfeil. Dies ist sinnvoll, wenn Stellen im Programm gesucht werden, an denen bestimmte Variablen evtl. ihren Wert verändern.

∆SUCHE_UL und *∆SUCHE_ULX* basieren dabei auf Informationen, die von ⎕MF geliefert werden.

2) Ausnutzung von ⎕SI für Debuging-Zwecke (Beispiel: ∆SPY_STMT)

Die Funktion *∆SPY_STMT* wird verwendet, um bei (z.B. durch einen Watchpoint) gestopptem Interpreter den Zustand der aktuellen Zeile in einer Form auszugeben, die für andere Kollegen (z.B. die Kollegen von der C++-Programmierung) verständlich sind und die wesentlichen Informationen zu dieser Programmzeile zusammenfassen.

- Durch Analyse des Status-Indikators ⎕SI erkennt ‘SPY_STMT, wo der Interpreter zur Zeit steht (blau markiert).

```
     SPY_STMT 0
⍝-----------------------------------------------------------------
⍝  FORM_DK_PGM[18]
⍝  ⍟
⍝  PRO_DK_PGM[41]
⍝  DK_PGMLPMU[8]
⍝  DK_PGM[72]
⍝  DK_PGM_U1[433]
⍝  DK_PGMU[459]
⍝  DK_HV[75]
⍝  ⍟
⍝  DK_FORMEL076[23] *
⍝-----------------------------------------------------------------
⍝ BWP←PX×((0<PX)×NX[X+M+1]-NX[X+T+1])÷NENNER
⍝-----------------------------------------------------------------
```

```
PX      ⍝  (54⍴829.87),0
NX      ⍝  932628077900036,26.932628077900036,25.99654 36,25.09027...
X       ⍝  31
M       ⍝  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  ...
T       ⍝  54
NENNER  ⍝  0.36629600000000192 0.35483099999999988 0.3437049999  ...
```

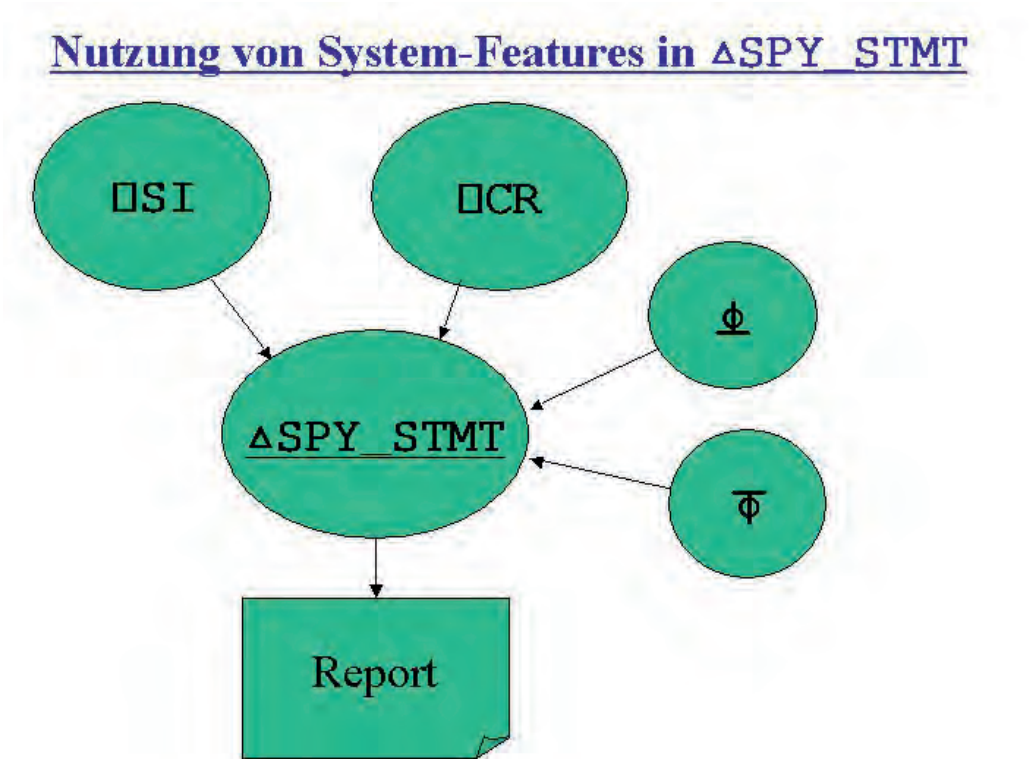


Abb. 3: Nutzung von System-Features in *∆SPY_STMT*

- Mit Hilfe von ⎕CR erhält man den Inhalt der aktuellen Zeile (rot markiert).

- Durch Analyse der aktuellen Zeile erhält man die verwendeten Variablennamen.

- Durch Aktivierung der verwendeten Variablennamen kommt man zu der gewünschten Darstellung der Variablen-Belegungen (grün markiert).

3) Manipulation der Variablen

⎕WATCHPOINTS

⎕WATCHPOINTS ist eine Steuerungs-variable des APL-Debuggers („Code -Walker"). Es handelt sich hierbei um eine (N x 2)- Matrix von Strings.

Spalte 1: Namen der überwachten Variablen.
Spalte 2: Boolsche Ausdrücke dazu.

Jedesmal, wenn einer Variablen der linken Spalte ein Wert zugewiesen wird, wird der zugehörige boolsche Ausdruck (rechte Spalte) ausgewertet. Ergibt sich ein „True" („1"), stoppt der Interpreter.

Normalerweise wird ⎕WATCHPOINTS über die Menueleiste des Session-Managers manuell verändert, es ist aber auch möglich, ⎕WATCHPOINTS von Programmen aus zu manipulieren.

Beispiel: ∆WATCHX

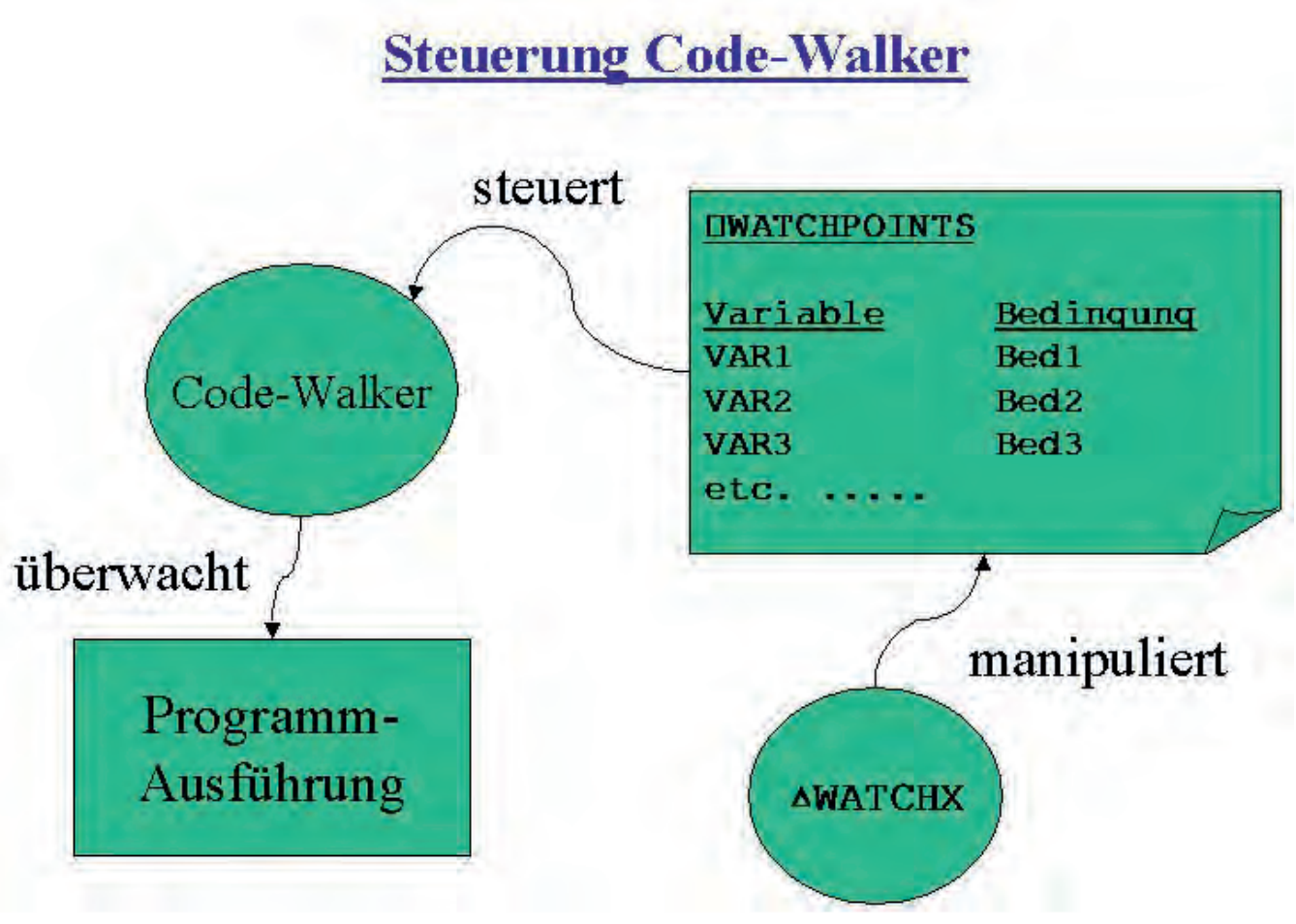


Abb. 4: Steuerung Code-Walker

∆WATCHX bekommt als Argument einen Variablennamen X und manipuliert die Tabelle ⎕WATCHPOINTS dann dergestalt, daß der Programmfluß beim nächsten Lauf stoppt, sobald der aktuelle Wert nach X zugewiesen wird. Durch geschickte Kombination von ∆SPY_STMT und ∆WATCHX kann man so einen „Quasi-Rückwärtslauf" des Interpreters realisieren.

Die Frage „Wo kommt der aktuelle Wert der Variablen X eigentlich her" läßt sich so in manchen Fällen relativ bequem beantworten.

Abb. 5: Quasi-Rückwärts_Lauf

4) Ausnutzung von ⎕COPY zur Untersuchung von Workspace-Unterschieden

Gegeben seien zwei fast gleiche Workspaces:

WS1: von letzter Woche

WS2: von heute

Der Ausdruck `y1←f(x)` in WS1 liefert ein anderes Ergebnis als `y2←f(x)` in WS2.

Aber eigentlich sollten beide Ergebnisse gleich sein. Geändert haben sich vielleicht 50 Funktionen und 25 Parameter-Tabellen – aber nur an Stellen, die EIGENTLICH überhaupt keinen Einfluss auf das Ergebnis haben sollten ...

WAS ALSO TUN ?

Vorgehensweise

- Nehme eine Kopie von Workspace WS1
- Kopiere mit ⎕COPY nacheinander alle verdächtigen Programme/Variablen aus WS2 hinein
- Untersuche jedesmal, ob sich das Ergebnis y1„f(x) verändert hat
- Mache das alles nicht manuell, sondern schreibe Dir eine Funktion dafür

Auch mit diesem sehr simplen Algorithmus lassen sich oft langwierige und komplizierte Fehler-Analysen vermeiden. Voraussetzung hierfür ist aber eine Funktion wie ⎕COPY, die es ermöglicht, während des Programmlaufes gezielt einzelne Unterfunktionen von f auszutauschen.

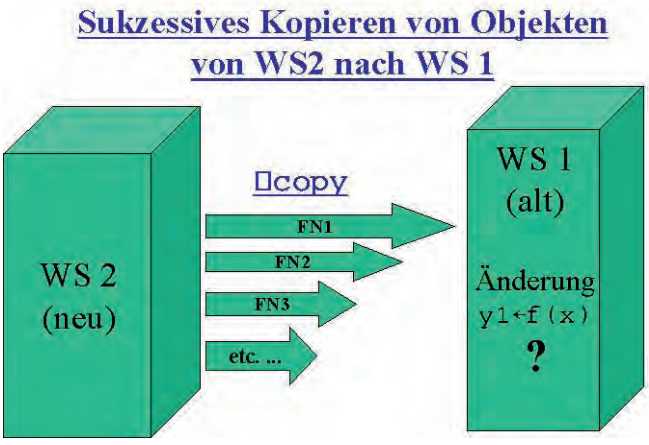


Abb. 6: Sukzessives Kopieren

Fernziel: Stärkere maschinelle Unterstützung durch komplexe Testwerkzeuge

Da sich bereits mit diesen einfachen „Tools" eine deutliche Erleichterung der täglichen Test-Arbeit erreichen läßt, liegt der Gedanke nahe, durch den Einsatz komplexerer Testwerkzeuge noch mehr Analyse-Arbeit an den Rechner abzugeben. Inwiefern diese Ideen realistisch sind und ob sie letztendlich ein befriedigendes Kosten/Nutzen-Verhältnis zeigen, muß die Zukunft erweisen.

Einige der Ideen seien hier nur kurz skizziert:

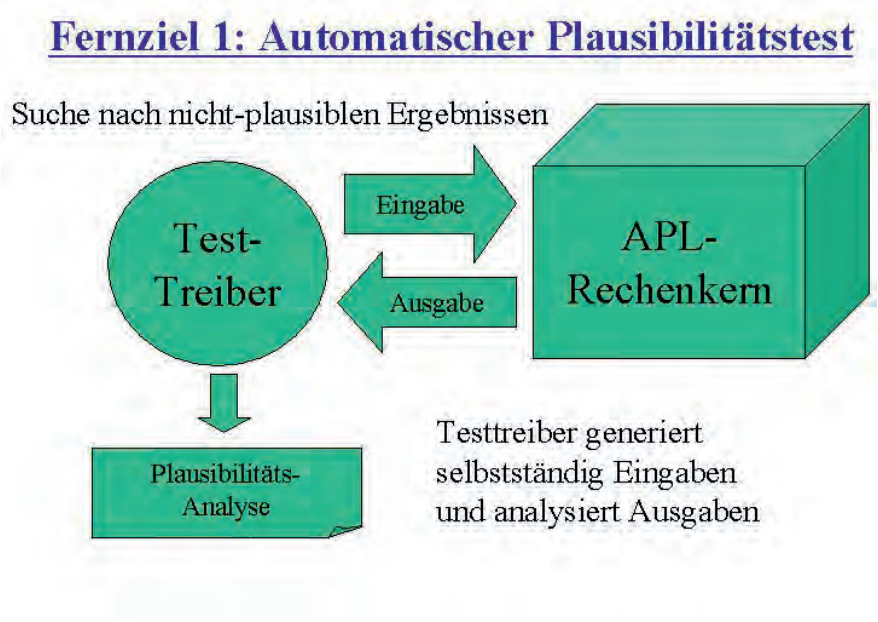


Abb. 7: Fernziel 1

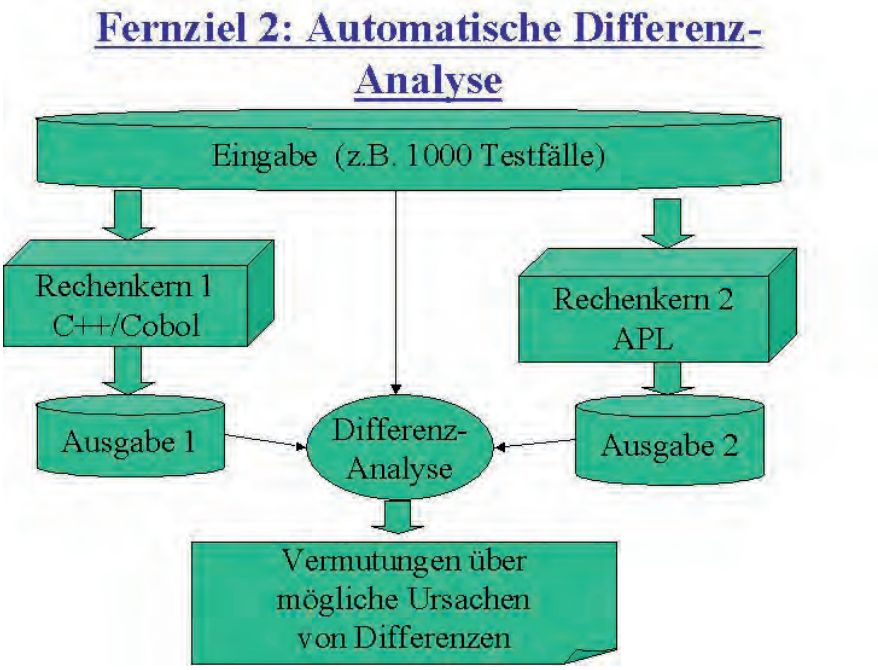


Abb. 8: Fernziel 2

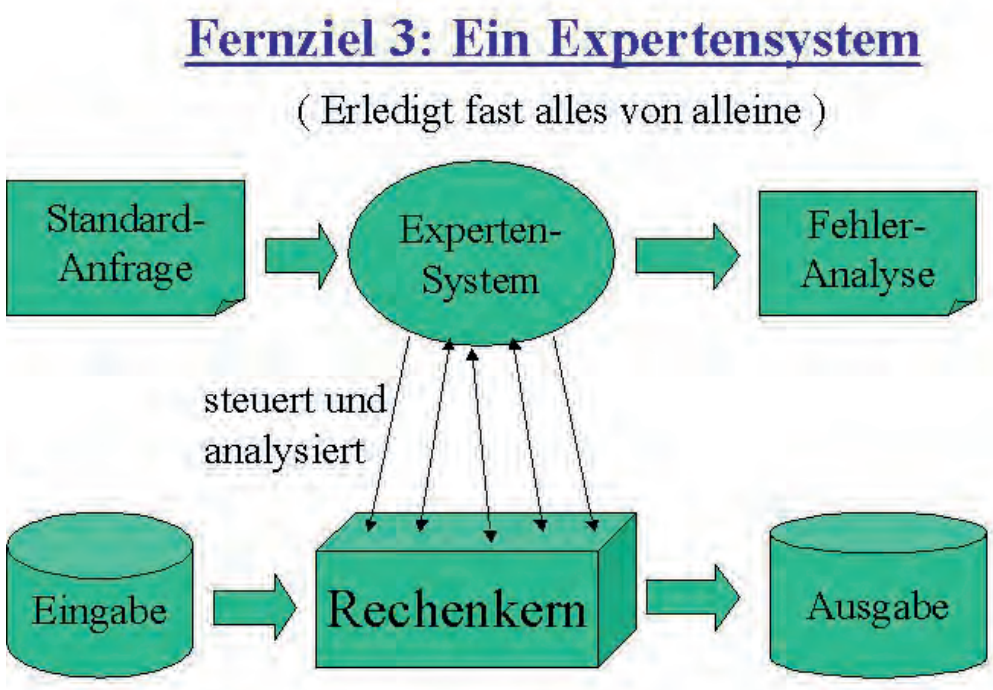


Abb. 9: Fernziel 3

Fazit

Durch geschickte Ausnutzung einiger weniger Features der Sprache APL läßt sich die Analyse komplexer mathematischer Algorithmen erheblich beschleunigen. Grundstrategie dabei ist es, immer wiederkehrende Arbeitsschritte im Analyseprozess zu automatisieren und die so geschaffenen Tools nach und nach zu immer leistungsfähigeren Werkzeugen zusammenzuschalten.

Einiges ist bereits erreicht, aber weitere Anstrengungen in diese Richtung sind notwendig, um mit der wachsenden Komplexität moderner Softwaresysteme einigermaßen Schritt halten zu können.

Kontakt zum Autor:

E-Mail: ralf.herminghaus@axa.de

APL-Journal

30. Jg. 2011, ISSN 1438-4531

Herausgeber: Dr. Reiner Nussbaum, APL-Germany e.V., Mannheim, http://www.apl-germany.de

Redaktion: Dipl.-Volksw. Martin Barghoorn (verantw.), TU Berlin, Franklinstr. 28, D-10587 Berlin, Tel. (030) 314 24392, Fax (030) 314 25901

Verlag: RHOMBOS-VERLAG, Berlin, Kurfürstenstr. 17, D-10785 Berlin, Tel. (030) 261 9461, Fax (030) 261 6300, eMail: verlag@rhombos.de, Internet: www.rhombos.de

Erscheinungsweise: halbjährlich

Erscheinungsort: Berlin

Satz: Rhombos-Verlag, Vicky Kletke

Druck: dbusiness.de GmbH, Berlin

Copyright: APL Germany e.V. (für alle Beiträge, die als Erstveröffentlichung erscheinen)

Fotonachweis Titelseite: Martin Müller / pixelio.de

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutzgesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürfen. Eine Haftung für die Richtigkeit der veröffentlichten Informationen kann trotz sorgfältiger Prüfung von Herausgeber und Verlag nicht übernommen werden. Mit Namen gekennzeichnete Artikel geben nicht unbedingt die Meinung des Herausgebers oder der Redaktion wieder. Für unverlangte Einsendungen wird keine Haftung übernommen. Nachdruck ist nur mit Zustimmung des Herausgebers sowie mit Quellenangabe und Einsendung eines Beleges gestattet. Überarbeitungen eingesandter Manuskripte liegen im Ermessen der Redaktion.

Allgemeine Informationen

(Stand 2011)

APL-Germany e.V. ist ein gemeinnütziger Verein mit Sitz in Düsseldorf. Sein Zweck ist es, die Programmiersprache APL, sowie die Verbreitung des Verständnisses der Mensch-Maschine Kommunikation zu fördern. Für Interessenten, die zum Gedankenaustausch den Kontakt zu anderen APL-Benutzern suchen, sowie für solche, die sich aktiv an der Weiterverbreitung der Sprache APL beteiligen wollen, bietet APL-Germany den adäquaten organisatorischen Rahmen.

Auf Antrag, über den der Vorstand entscheidet, kann jede natürliche oder juristische Person Mitglied werden. Organe des Vereins sind die mindestens einmal jährlich stattfindende Mitgliederversammlung sowie der jeweils auf zwei Jahre gewählte Vorstand.

1. Vorstandsvorsitzender
Dr. Reiner Nussbaum
Dr. Nussbaum gift mbH, Buchenerstr. 78, 69259 Mannheim,
Tel. (0621) 7152190.

2. Vorstandsvorsitzender:
Martin Barghoorn
Sekr. FR 6-7, Technische Universität Berlin, Franklinstr. 28,
10587 Berlin,
Tel. (030) 314 24392

Schatzmeister
Uwe Janza
Janza Software Consulting,
Mittlere Kanalstraße 4a
90429 Nürnberg
Tel. 0911-327075,
Fax -327076

Beitragssätze

Ordentliche Mitglieder:

Natürliche Personen	32,- EUR*
Studenten / Schüler	11,- EUR*

Außerordentliche Mitglieder:

Jurist./natürl. Pers.	500,- EUR*

* Jahresbeitrag

Bankverbindung
BVB Volksbank eG Bad Vilbel
BLZ 518 613 25
Konto-Nr. 523 2694

Hinweis:
Wir bitten alle Mitglieder, uns Adressänderungen und neue Bankverbindungen immer sofort mitzuteilen. Geben Sie bei Überweisungen den namen und/oder die Mitgliedsnummer an.

Einzugsermächtigung

Mitglieds-Nr.: _____

Ich erkläre mich hiermit widerruflich damit einverstanden, daß APL Germany e.V. den jeweils gültigen Jahres-Mitgliedsbeitrag von meinem unten angegebenen Konto abbucht. Mit der Datenübermittlung an das oben genannte Kreditinstitut bin ich einverstanden. Einen eventuell bestehenden Dauerauftrag habe ich bei meiner Bank gelöscht.

Bankbezeichnung: _____

BLZ: _____ Konto-Nr.: _____

Datum: _____ Unterschrift: _____