RUHR - UNIVERSITÄT BOCHUM

Arbeitsberichte

des

Rechenzentrums

Direktor: Prof. Dr. H. Ehlich

Nr. 7210 BO.E6.03

BOGOL-STRING, eine flexible Zeichenkettenverarbeitung in ALGOL 60

von

Manfred Rosendahl

Bochum, Oktober 1972
Universitätsstraße 150 Gebäude NA

Inhaltsverzeichnis

0. Einleitung

Die Algol 60 Implementierung des TR 440 [1] ist eine sehr weitgehende und vollständige Implementierung der Sprache Algol 60 [2]. Die Sprache Algol 60 enthält jedoch keine Anweisungen zur Stringbearbeitung. Strings sind zwar als Daten im Sprachumfang enthalten, können jedoch nur ein- und ausgegeben und als formale Parameter übergeben werden. Lediglich für sehr beschränkte Vergleiche steht die Prozedur EQUIV zur Verfügung.

Als eine zur Zeichenkettenverarbeitung sehr geeignete Sprache gilt SNOBOL. Gebräuchlich sind die beiden Versionen SNOBOL 3 [3] und SNOBOL 4 [4].

Im Rahmen des Programmiersystems BOGOL-STRING werden die meisten Möglichkeiten von SNOBOL 4 in Algol-Programmen ansprechbar.

BOGOL-STRING ist ein Teil des Systems BOGOL (BOchumer-alGOL) [1)], das zur Erweiterung der Algol Sprachmöglichkeiten gedacht ist.

---

1) Weitere Teile des BOGOL-Systems:

(1) BOGOL-TAS, zur systemnahen Programmierung in Algol (bereits implementiert) [5].

In Vorbereitung:

(2) BOGOL-LISP. Hier wird Algol-Programmen eine LISP-Datenstruktur beigegeben. Dies erlaubt es, LISP [6] und AIDA [7, 8] - Funktionen in Algol zu benutzen.

(3) BOGOL-FORMAC, zur Formelmanipulation und zum symbolischen Rechnen.

1. Überblick über das System BOGOL-STRING

1.1. Aufbau des Systems

Das System BOGOL-STRING ist als Unterprogrammsystem aufgebaut. Die beiden hauptsächlichen Prozeduren sind MAT und ASS.

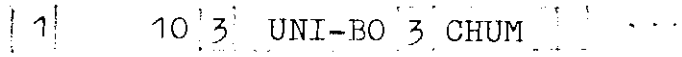
MAT (A, B_1, ..., B_n) testet einen String A auf Bedingungen B_1, ..., B_n. Das Ergebnis kann nachher mit den Prozeduren SUCC oder FALL abgefragt werden.

ASS (A, C) ersetzt String A durch String C. Dies kann auch von Testbedingungen abhängig gemacht werden:

ASS (A, B_1, ..., B_n, C).

Die Darstellung der Strings ist die gewöhnliche Algol-TR 440 Stringdarstellung, also:

1. Ganzwort:	Länge des Strings mit TK1,
2. und folgende Ganzworte:	Stringinhalt mit 6 Zeichen/Wort.
Beispiel:	UNI-BOCHUM



Ein solcher String wird in aufeinanderfolgenden Zellen i.A. einem Array abgespeichert.

Ein so abgespeicherter String kann wie ein gewöhnlicher Algol-string behandelt werden, also als Parameter auftreten oder ausgegeben werden.

Er entspricht jedoch nicht dem A-Format der Knuth-Prozedur INPUT bzw. OUTPUT.

1.2. Deklarationen und Handhabung

Die BOGOL-STRING Prozeduren werden wie gewöhnliche Algol code-Prozeduren in der Form:

< type > procedure NAME(X); code;

deklariert. Wegen der Typenkennung ist jedoch folgendes zu beachten:

Wird einer integer-Größe eine real-Größe zugewiesen, so erfolgt eine Rundung. Dies läuft jedoch für TK $\neq$ 0 auf einen Alarm. Daher werden Funktionsprozeduren als integer procedure deklariert.

Die Prozeduren des BOGOL-Systems liegen als Montageobjekte in der Bibliothek BOGOL auf der LFD (langfristige Datenhaltung) vor.

Nach dem Kommando ⌷ BIBANM., BOGOL sind sie wie gewöhnliche Algol-Prozeduren der &OEFDB (öffentliche Datenbasis) zu benutzen.

Zur Vermeidung des lästigen Deklarierens der code-Prozeduren ist ein Precompiler geplant, der in einem Algol-Programm fehlende Deklarationen einsetzt.

2. Strings

2.1. Parametertypen der STRING-Prozeduren

Bei den Stringprozeduren können mehr als die gewöhnlichen Algol 60 Parametertypen vorkommen.

In den Prozedurbeschreibungen werden für die erlaubten Typen der aktuellen Parameter folgende Bezeichnungen verwandt:

string	String
array	Feldname
ref	Referenz (Adresse) einer indizierten Variablen. Dies kann erreicht werden durch Angabe der Variablen, wenn ihr Inhalt TK1 (Kopfwort) ist, oder durch einen Ausdruck, der die Adresse angibt, z.B. durch Prozedur REF(BOGOLTAS [5]).
integer	Algol-Variable oder Größe mit integer-Wert.
real	Algol-Variable oder Größe mit beliebigem Wert.
pattern	string, Feldname oder Ausdruck für einen errechneten String (z.B. mit CAT (Verkettung) oder IND (Indirekte Referenz));
expression	Ausdruck für einen errechneten String (z.B. durch CAT oder IND).
var	Variable, keine Konstante oder Ausdruck.

2.2. Umformung Zahlen :=: Strings

Um Strings in Zahlen zu wandeln, dient:

```
real procedure NUMV (A);
pattern A;
```

Das Ergebnis ist der Zahlenwert des Strings A.

Die Umkehrung liefert:

```
string procedure STRV (A);
real A;
```

Das Ergebnis ist der String, der die Zahl A darstellt.

Die Strings werden dabei in der Algol-Syntax eingegeben. Das String-Resultat liefert die signifikanten Stellen der Zahl. Zu große oder zu kleine Zahlen werden in Gleitkommadarstellung gebracht.

2.3. Ein- und Ausgabe von Strings

Die Strings entsprechen genau den gewöhnlichen Algolstrings, daher können sie mit PRINT oder OUTARRAY ausgegeben und in Stringklammern mit PRINT bzw. INARRAY

eingegeben werden.
Als spezielle String-EA dienen die Prozeduren EIN und AUS.

```
procedure EIN(A);
array oder ref A;
```

In der Normaleingabe (Fremdstring bei Abschnitt, Konsoleingabe bei Gespräch) wird eine Zeile gelesen und nach A gespeichert.

```
procedure EIN(A, G);
array oder ref A;
integer G;
```

Wie oben, jedoch Gerätenummer G.

```
procedure EIN(A, G, S);
array oder ref A;
integer G, S;
```

Vom Gerät G wird Satznummer S eingelesen. Die zugehörende Datei muß natürlich ein Random (RAN oder RAM)-Datei sein.

Die analogen Aufrufe zur Ausgabe sind:

```
procedure AUS(A);
array oder ref oder pattern A;
```

Der mit A bezeichnete String wird auf den Normalausgabemedien (Druckerprotokoll im Abschnitt, Konsolprotokoll im Gespräch) ausgegeben.

Mit Geräte- und Satznummer sind die entsprechenden Aufrufe:

AUS(A, G) und
AUS (A, G, S).

Vor der Ausgabe wird jeweils eine neue Zeile begonnen, so daß jeder Aufruf von AUS eine Zeile ergibt.

3. "Pattern-matching"

3.1. "Matching"-Prozedur MAT

Diese Prozedur zur Prüfung des Auftretens bestimmter Teilstrings in einem String (das "pattern-matching") ist die Hauptprozedur zur String-manipulation.

Die Prüfung kann in den Prozeduren:
MAT (nur Prüfung) und
ASS (Prüfung mit Zuweisung (3.2.)) erfolgen.

Die "pattern-matching"-Prozedur hat folgendes Format:

MAT (A, B_1, ..., B_n);

Bei A wird der zu untersuchende String angegeben. Dies kann sein:

array: Der String steht auf dem Feld, vom 1. Element an.

string: Es wird der String untersucht, der unter dem Namen des strings auf Parameter A abgespeichert wurde (Indirekte Referenz).

str.-expr: Der errechnete String wird untersucht.

Die Positionen B_1, ..., B_n können verschieden besetzt sein.

Diese Parameter spezifizieren gewisse Teilstrings oder Bedingungen, die fortlaufend im String A vorkommen bzw. erfüllt sein müssen, damit der "pattern-match" erfolgreich ist. Dieser Erfolg kann nachher abgefragt werden (SUCC, FALL). Dieses "matching" testet ein "scanner"-Algorithmus,der die einzelnen Bedingungen überprüft. In einigen Fällen können mehrere evtl. verschieden lange Teilstrings die Bedingung erfüllen. Dann wird zunächst die erste Alternative (bei ALT) bzw. der kürzeste mögliche Teilstring (evtl. auch der Nullstring) genommen. Kann nachher eine spätere Bedingung nicht erfüllt werden, so wird wieder zurückgegangen und eine andere Alternative oder ein längerer Teilstring genommen.

Der "pattern-matching" Algorithmus verläuft genauso wie in SNOBOL nach folgenden Regeln:

Zunächst im Mode "unanchored".

(1) Es wird versucht, die erste Bedingung (z.B. ein String) vom 1. Zeichen des zu untersuchenden Strings A an zu matchen. Geht dies nicht, dann vom 2. Zeichen an und so weiter.

(2) Es werden nach rechts fortlaufend alle folgenden Bedingungen durch aufeinander folgende Teilstrings von A zu "matchen" versucht.

(3) Kann eine Bedingung nicht erfüllt werden, so wird versucht, die vorhergehende Bedingung durch eine andere Alternative (bei Prozedur ALT) oder einen längeren Teilstring zu "matchen".

(4) Hat die vorhergehende Bedingung keine freie Alternative mehr bzw. kann die Länge des "matchenden" Teilstrings nicht vergrößert werden, so wird weiter nach links gegangen und die davor liegende Bedingung untersucht usw.

Der "matching"-Algorithmus läuft also hin und her bis entweder

(a) die letzte Bedingung "gematcht" wurde, dann ist der "pattern match" erfolgreich (SUCC) oder

(b) die erste Bedingung mangels freier Alternativen oder nicht mehr zu verlängerndem Teilstring nicht "gematcht" werden kann. Dann war der "pattern match" ohne Erfolg (FAIL).

Anschließend liefern die boolschen Prozeduren SUCC und FAIL die entsprechenden Werte.

Im Mode " anchored ", einstellbar durch ANCHOR(1), muß die erste Bedingung durch einen Teilstring, der am Anfang des zu untersuchenden Strings liegt,

"gematcht" werden.

Nach Aufruf der Initialisierungsprozedur SNOBOL ist der Mode "unanchored" eingestellt.

Die Parameter B_1 bis B_n können annehmen:

integer 0	: Der Teilstring kann ein beliebiger string beliebiger Länge sein. (SNOBOL 4 ist dies ARB).
integer n > 0	: Teilstring beliebiger String der der Länge n.
string	: Teilstring ist der angegebene String.
Str.-expr.	: Teilstring ist der errechnete String.
array	: Teilstring ist der auf dem Array abgelegte String.
ref	: Teilstring ist der String auf den verwiesen wird bzw. der dort beginnt.

Beispiel:

```
MAT(A,'('LAND')', 0, '('WASSER')');
```

Sei auf A: '('LAND UND WASSER')'

so wird für 0 der String: '(' UND ')' genommen.

Dagegen würde:

MAT (A, '('LAND')', 3, '('WASSER')');

nicht "matchen", da '('␣UND␣')' nicht die Länge 3 hat.

Außerdem können auf den Parameterpositionen $B_1,\ldots,B_n$ Aufrufe von "matching"-Prozeduren stehen (m-procedure). Dies sind die Prozeduren:

ALT

SPAN und BREAK

ANY und NOTANY

TAR, RTAB und REM

POS und RPOS

ARBNO und BAL

Die m-procedure werden als integer-procdure deklariert. Die Funktionswerte, die sie liefern, dienen als Steuerung für den "matching"-Algorithmus in der Prozedur MAT.

Außerdem die Prozeduren, die ref oder Str.-expr. ergeben. Diese werden in Kapitel 2 beschrieben.
Dazu noch die Prozeduren zur Zuweisung an "string-Variablen" : SV und IV.

Kommt in den Parametern $B_1,..,B_n$ als Prozeduraufruf nur SV vor und sonst nur die angeführten Möglichkeiten,

so kann statt der Prozedur MAT auch die Prozedur MAT3 benutzt werden (= SNOBOL 3). Diese läuft dann etwa 25 % schneller als MAT.

Für den "scanner"-Algorithmus gibt es zwei verschiedene Modes. Der normale Mode ist der "unanchored mode". Der "matchende" Teilstring kann an beliebiger Stelle im String beginnen.

Im "anchored mode" muß er jedoch mit dem 1. Zeichen des Strings beginnen. Durch die Prozedur ANCHOR kann der Mode umgestellt werden.

Beispiel:
```
ANCHOR(0) "unanchored"
MAT(A, B, C, D);
ist aquivalent zu
ANCHOR(1) "anchored"
MAT(A, 0, B, C, D);
```

3.2. Ersetzungs-Prozedur: ASS

Die Ersetzungsprozedur ASS dient dazu, einem Array oder Teilarray einen String zuzuweisen. Außerdem kann eine indirekte Referenz aufgebaut werden. Dies kann auch in Abhängigkeit eines "matching"-Tests gemacht werden.

3.2.1. Ersetzung ohne "matching"

Der Aufruf hat die Form:

```
ASS (A, C);
```

Dabei wird der String C auf der Stringvariablen A abgelegt. Ist A array oder ref, so wird ab der durch A bezeichneten Adresse abgelegt. Ist A ein String, so wird dieser als indirekte Referenz verstanden.

Also statt: `ASS(IND('('ABC')',C);` kann man kurz
`ASS('('ABC')',C);` schreiben.

Der Parameter C bezeichnet einen String. C kann sein: string, array, ref oder expression, also ein beliebiges pattern.

Beispiel:
```
array A [1:10];
ASS(A,'('RECHENZENTRUM')');
```

Der String wird auf A abgelegt.

3.2.2. Ersetzung mit "matching"

Die Ersetzungsprozedur ASS kann vor der Ersetzung wie die Prozedur MAT einen "matching"-Test durchführen. Die Ersetzung hängt dann von dem Testergebnis ab. Der Aufruf hat die Form:

ASS(A, B_1,..., B_n, C);

Zunächst wird wie beim Prozeduraufruf

MAT (A, B_1, ..., B_n)

ein Matching-Test durchgeführt. Ist dieser nicht erfolgreich, so bewirkt die Prozedur ASS keine Ersetzung. Ist der Test jedoch erfolgreich und die Elemente a_i, a_{i+1}, ..., a_j "matchen" die Bedingungen B_1, ..., B_n, so erhält A die Gestalt: $a_1,\ldots,a_{i-1},c_1,\ldots,c_k,a_{j+1},\ldots,a_n$. Also $a_i,\ldots,a_j$ werden durch C = $c_1\ldots c_k$ ersetzt. Die Möglichkeiten für die Parameter sind die gleichen wie bei den Prozeduren MAT bzw. ASS.

Unter den gleichen Bedingungen wie MAT3 kann auch die Prozedur ASS3 benutzt werden, die den "matching"-Test schneller durchführt.

3.3. Wertzuweisung während des "matching"-Algorithmus

3.3.1. Bedingte Wertzuweisung: SV

Tritt in den Spezifikationsparametern B_1 bis B_n des "matching"-Algorithmus eine Parameterfolge:

...,B_i, SV(D),...

auf, so wird, falls der gesamte "Match" erfolgreich ist, der Teilstring, der die Spezifikation B_i "gematched" hat, auf die Stringvariable D zugewiesen. D kann also array oder ref sein.

Beispiel: array A, D 1:10 ;
MAT (A, '('E')', 0, SV(D), 0, '('E')');

Der erste Teilstring, der zwischen zwei 'E' steht, wird auf D zugewiesen. Jedoch nur, falls das 2. 'E' gefunden wird.

3.3.2. Unbedingte Wertzuweisung: IV

Tritt die Parameterfolge

..., B_i, IV(D), ...

auf, so wird die Wertzuweisung auf jeden Fall vorgenommen, falls B_i "matched", unabhängig ob die späteren Bedingungen erfüllt werden.

3.4. Alternativen ALT

Diese Prozedur hat die Gestalt:

m-procedure ALT (A_1, A_2, ..., A_n)
pattern, m-procedure oder integer A_i.

Zur Zeit kann die Prozedur ALT jedoch noch nicht rekursiv aufgerufen werden.

Sollen für eine "matching"-Bedingung mehrer Alternativen bestehen, so kann dies durch die Prozedur ALT angegeben werden. Die verschiedenen Alternativen sind die Parameter der Prozedur ALT. Sie werden der Reihe nach getestet. Ist eine erfüllt, wird im Algorithmus weitergegangen. Kann später der "matching"-Algorithmus eine Bedingung nicht erfüllen, so wird zurückgegangen, und es werden die übrigen Alternativen untersucht. Die Parameter von ALT können alle Werte sein, die auch sonst für die Spezifikationen B_i erlaubt sind, z.Z. mit Ausnahme eines rekursiven Aufrufes von ALT selbst.

Beispiel: array A [1:10];

```
MAT(A, ALT('('ALT')', '('NEU')'), ALT('('DORF')', '('STADT')'));
```

Erfolgreich "gematcht" werden die Strings:

ALTDORF
NEUDORF
ALTSTADT
NEUSTADT

3.5. ANY und NOTANY

Diese Prozeduren haben die Gestalt:

```
m-procedure ANY(A);
  pattern (A);
m-procedure NOTANY(A);
  pattern (A);
```

Die Prozedur ANY "matcht" einen String aus einem Zeichen, das mit einem Zeichen des pattern A übereinstimmt.

NOTANY dagegen "matcht" einen String aus einem Zeichen, das mit keinem Zeichen des pattern A übereinstimmt.

Beispiel:
```
array A, B [1:10 ];
ASS (B, '('AEIOU')');
MAT(A,ANY(B),NOTANY(B),ANY(B));
```

A wird "gematcht", wenn es in A eine Folge Vokal, Konsonant, Vokal gibt.

3.6. SPAN und BREAK

Die Prozeduren haben die Form:

```
m-procedure SPAN(A);
  pattern A;
m-procedure BREAK(A);
  pattern A;
```

SPAN(A) "matcht" den String maximaler Länge, der nur aus Zeichen besteht, die in A vorkommen. BREAK(A) dagegen den String maximaler Länger, der kein Zeichen aus A enthält.

Der "match" ist bei beiden Prozeduren immer erfolgreich, evtl. wird nur der Nullstring "gematcht".

Beispiel:

```
ASS (B, '('12 ... 9')');
ASS (C, '('ABC ... Z')');
MAT (A, ANY(C), SPAN(CAT(B, C)));
```

Es wird in A der erste auftretende Identifier "gematcht".

3.7. TAB, RTAB und REM

Durch diese Prozeduren wird der Teilstring von der derzeitigen Cursor-Position bis zu einer bestimmten Stelle, von links bzw. rechts gezählt, "gematcht".

```
m-procedure TAB(N);
  integer N;
```

TAB(N) "matcht" von der derzeitigen Corsorposition bis einschließlich des N-ten Zeichen des Untersuchungsstrings.

```
m-procedure RTAB(N);
  integer N;
```

RTAB(N) "matcht" von der derzeitigen Cursorposition bis ausschließlich des N-ten Zeichens von rechts des Untersuchungsstrings.

Beispiel:

```
ASS(A,'('BOGOL440')');
MAT(A, 2, TAB(5));
```

TAB(5) "matcht" 'GOL'

```
MAT(A, 2, RTAB(3));
```

RTAB(3) "matcht" ebenfalls 'GOL'

m-procedure REM;

REM besitzt keinen Parameter und entspricht RTAB(0), d.h. "matcht" gerade den Reststring bis zum Ende.

3.8. POS und RPOS

Mit Hilfe der Prozeduren POS und RPOS kann die derzeitige Cursorposition getestet werden. Sie "matchen" jeweils den Nullstring, wenn die Cursorposition deren Parameter entspricht.

m-procedure POS(N);

integer N;

POS(N) "matcht" den Nullstring, wenn der Cursor zwischen dem N. und dem N+1 - Zeichen steht.

Beispiel:

(a) MAT(A, POS(0), SPAN'(' ')', POS(5));

MAT ist nur erfolgreich, wenn der String A mit genau 5 Blanks beginnt.

(b) MAT(A, POS(0), ANY('(' ')'));

MAT ist hier erfolgreich, wenn A mit wenigstens einem Blank beginnt.

POS(0) hat die gleiche Wirkung wie ein Durchsuchen im "anchored mode", also mit vorgestelltem ANCHOR(1);

m-procedure RPOS(N);
integer N;
RPOS(N) matcht den Nullstring, wenn sich der Cursor zwischen dem N. und dem N+1 - Zeichen von rechts befindet.

Beispiel: MAT(A, POS(0), B, RPOS(0));

MAT ist hier nur erfolgreich, wenn die Strings A und B übereinstimmen.

3.9. ARBNO

m-procedure ARBNO(A);
pattern A oder integer A;

ARBNO(A) matcht jeden Teilstring, der aus einer beliebigen Wiederholung des pattern A besteht. Der "match" ist in jedem Fall erfolgreich, da der leere String als 0-faches Auftreten von A erlaubt ist. Ähnlich wie bei "matchen" eines beliebigen strings wird auch hier zunächst der Nullstring versucht und erst wenn dies nachher nicht mehr fortzusetzen ist, werden auch die höheren

Wiederholungen getestet. Es ist also eine Testprozedur mit impliziten Alternativen.

Falls der Parameter A eine Zahl ist, "matcht" ARBNO(A) jeden Teilstring der Länge n*A mit n=0,1,2,... .

Beispiel: MAT(A, POS(0), ARBNO(2), RPOS(0));

"matcht" jeden geraden String A.

Der "match" ist bei ARBNO in jedem Fall erfolgreich, evtl. mit dem Nullstring.

3.10. CURS

m-procedure CURS(A);
var A;

Die Prozedur CURS "matcht" immer den Nullstring, hat also für den "matching"-Algorithmus keine Bedeutung. Es wird jedoch die aktuelle Cursorposition auf der Variablen A abgelegt.

Beispiel: array A [1:10] ; real B;
MAT(A, POS(0), SPAN('(' ')'), CURS(B));

Die Anzahl der führenden Blanks wird auf B abgelegt.

4. Stringfunktionen

Im Rahmen der gewöhnlichen Algol type-procedures können Funktionen nur einfache Zahlen- und logische Werte als Ergebnis liefern. Im Rahmen des STRING-Systems sollen jedoch auch Strings als Funktionsergebnis geliefert werden können. Intern wird dies so gehandhabt, daß die Adresse des Kopfwortes mit TK2 übergeben wird.

In den STRING-Prozeduren wird ein Wert mit TK2 entsprechend ausgewertet.

4.1. Concatenation CAT

string procedure CAT (A_1, A_2, ..., A_n);
pattern A_1, A_2, ..., A_n;

Die einzelnen Zeichenketten A_1 bis A_n werden verkettet. CAT darf nicht rekursiv aufgerufen werden.

Beispiel: ASS(A, CAT(A, '('END')'));

An den String A wird der String 'END' angehängt.

4.2. Indirekte Referenz IND

Bisweilen möchte man einen String ansprechen, dessen Namen wiederum als String und nicht als Algol-Variable angegeben ist. Dazu dient:

```
string procedure IND(A);
pattern A;
```

Das Ergebnis der Prozedur ist der unter dem pattern A abgelegte String.

```
ASS(A, '('ASTRING')');
ASS(IND(A), '('ASTRING STRING')');
```

ASS(B,A) 'ASTRING' wird auf B abgelegt.
ASS(B, IND(A)); 'ASTRING STRING' wird auf B abgelegt.

Die indirekte Referenz kann auch mehrfach benutzt werden. Falls in der Prozedur ASS der 1. Parameter ein String bzw. das Resultat einer Stringfunktion ist, wird dieser String als Name des zu untersuchenden bzw. zu ändernden Strings genommen.

Beispiel:
```
ASS('('A')', '('ASTRING')');
ASS('('ASTRING')','('ASTRINGSTRING')');
IND('('A')') liefert 'ASTRING'
IND(IND('('A')')) liefert 'ASTRINGSTRING'
```

4.3. Stringarrays ARR

Etwa zur Ablage von Tabellen sind ein- oder mehrdimensionale Stringarrays praktisch. Um solche zu erstellen und zuzugreifen, dient die Prozedur ARR.

Falls in Algol ein array A [1:N, 1:M] definiert wird, so wird folgendermaßen abgelegt:

A(1,1) A(2,1) A(3,1) ... A(N,1) A(1,2) ... A(N,2) ... A(N,M)

Die Ablage erfolgt also spaltenweise. Für höhere Dimensionen gilt ebenso, daß bei der Ablage die vorderen Indizes schneller laufen.

Möchte man also einen Stringarray A mit den Grenzen:

$$[\, a_1 : b_1 \;,\; a_2 : b_2 \;,\; \ldots\ldots \; a_n : b_n \,]$$

deklarieren, so muß A wie folgt deklariert werden:

$$\underline{\text{array}}\ A\ [\, a_o : b_o \;,\; a_1 : b_1 \;,\; \ldots\ldots \; a_n : b_n \,]$$

Die Differenz $b_o - a_o$ bestimmt die mögliche Länge der Stringelemente in A. Mögliche Anzahl der Zeichen = $= (b_o - a_o - 1) \cdot 6$.

procedure ARR (A, I_1, ..., I_n);
value I_1, ..., I_n ; integer I_1, ..., I_n ; array A;

ARR (I_1, ..., I_n) liefert dann die Adresse des Elements A [a_o, I_1, ..., I_n] mit TK2 als Stringadresse.

4.4. Benutzerdefinierte Stringfunktionen

Es können auch Stringfunktionen definiert werden. Dabei muß nur die Konvention beachtet werden, daß die Kopfwortadresse mit TK2 als Funktionswert übergeben wird. Für die Operationen wie Typenkennungsetzen, Adressrechnung, Zeichenverarbeitung, siehe BOGOL-TAS [5]. Da die lokalen Variablen einer Prozedur beim Verlassen der Prozedur ihre Gültigkeit verlieren,muß das Feld, auf dem der String abgelegt wird, own, d.h. im TR440-Algol, auf einem zur Prozedur gehörenden common-Feld abgelegt werden.

Als Beispiel die Prozedur CAT (nur für Strings):

```
common CAT
array CATSTR [ 0:100 ] ;
real procedure CAT;
begin
PARV(1,A,AB);
PARV(2,B,BB);
```

```
AZR := REF(CATSTR [0]);
AS(AZR, AD(A,B));
TOK(AB, F6, AZR, F6, SB(A, FK(1))));
TOK(BB, F6, AZR, AD(F6, A), SB(A, FK(1)));
ZT2 (AZR);
CAT := AZR;
end;
```

Bezüglich der Prozeduren PARV, REF, TOK, AD, SB, FK, ZT2, siehe BOGOL-TAS [5].

5. Zusätzliche primitive Funktionen

5.1. SIZE

```
integer procedure SIZE(A);
pattern A;
```

Die Funktion SIZE liefert die Länge des Strings A;

5.2. REPLACE

```
procedure REPLACE (X, Y, Z);
pattern X, Y, Z;
```

Im String X wird jedes Zeichen, das in Y vorkommt, durch das an gleicher Stelle vorkommende Zeichen in Z ersetzt. Falls die Länge von Y und Z unterschiedlich oder 0 sind,

wird keine Änderung vorgenommen, dafür liefert die Funktion FAIL nachher true.

Beispiel: ASS(A, '('01101')');
REPLACE (A, '('01')', '('10')');

liefert das 1'er Kompliment.

5.3. Trim

Die Prozedur TRIM entfernt von einem String die Blanks am Anfang und am Ende.

procedure TRIM (A);
pattern A;

Der Parameter A darf jedoch kein explizieter String sein, da dieser schreibgeschützt abgelegt wird.

5.4. DUPL

procedure DUPL (A, N);
pattern A; integer N;

Der String A wird so oft an sich selbst angefügt, bis er N-mal vorhanden ist.

Beispiel: ASS(A, '(' ')');
DUPL(A, 50);
erzeugt einen String aus 50 Blanks.

5.5. IDENT und DIFFER

IDENT und DIFFER sind logische Funktionen, die zwei Strings auf Identität vergleichen.

```
boolean procedure IDENT (X,Y);
pattern X, Y;
boolean procedure DIFFER (X,Y);
pattern X,Y;
```

IDENT liefert true, wenn X und Y gleich sind, DIFFER umgekehrt.

5.6. CAS

```
procedure CAS (A, B1, ..., Bn);
pattern A, B1, ..., Bn;
```

Die Prozedur CAS vereinigt Concatanation (CAT) und Zuweisung (ASS). An den String A werden die Strings B_1 bis B_n angehängt.

6. Testmöglichkeiten

Zur Überwachung des Programmablaufs gibt es eine eingebaute Tracemöglichkeit. Ebenso kann eine vom Benutzer geschriebene Traceprocedur über Systemparameter eingeschaltet werden. Desweiteren existieren Dumpmöglichkeiten.

6.1. TRACE

Der Systemtrace wird durch den Prozeduraufruf TRACE(1) eingeschaltet und durch TRACE(0) ausgeschaltet.

Beispiel: A = 'HAUS UND LAND UND WASSER'

```
Statement:  MAT(A, '('HAUS')', 0, SV(B), '('LAND')');
liefert:    MAT: [HAUS] *[ UND ] * [LAND] UND WASSER
```

Die einzelnen "matchenden" Teilstrings werden in eckigen Klammern gedruckt.

Kann eine Bedingung nicht "gematcht" werden, erfolgt der Ausdruck /// und am Ende der Zeile *F*.

Bei einer Zuweisung, z.B. ASS(A, '('LAND UND WASSER')'); wird der zugewiesene String gedruckt:

```
ASS: LAND UND WASSER
```

Folgt vor der Zuweisung noch ein "matching"-Test, wird beides ausgedruckt:

```
Statement: ASS(A, '('UND')', '('-')');
MAT:       LAND  [UND]  WASSER
ASS:       LAND  -   WASSER
```

6.2. DUMP

Mit dieser Prozedur kann ein Dump von Stringinhalten bewirkt werden. Die verschiedenen Aufrufformen sind:

DUMP (N_1, A_1, N_2, A_2, ...);
pattern A_1, ..., A_K;
string N_1, ..., N_K;

Die Strings A_1, ..., A_K werden ausgedruckt. Als Stringnamen werden zuvor jeweils N_1, ..., N_K gedruckt.

DUMP (1, N_1, A_1, N_2, A_2, ...);
Die jeweiligen Strings A_i werden unter ihrem Namen N_i in den Dumpvektor aufgenommen.

DUMP(0, N_1, A_1, N_2, A_2, ...,);
Die Strings werden im Dumpvektor gelöscht.

DUMP;
Alle im Dumpvektor enthaltenen Strings werden ausgedruckt.

DUMP(0);
Alle folgenden Dumpbefehle sind unwirksam; mit Ausnahme von DUMP(1);

DUMP(1);
Alle Dumpbefehle sind wieder wirksam.

DUMP(-1);
Der Dumpvektor wird gelöscht.

Nach Normierung des Systems (siehe Prozedur SNOBOL) sind die Dumpbefehle unwirksam.

6.3. Benutzerdefinierte Testmöglichkeiten

Mit Hilfe der Prozedur AUS können Testausdrucke programmiert werden. Ob die Ausdrucke erfolgen sollen, kann durch die Variablen TRACEV, FTRACEV, DUMPV abgefragt werden.

7. Programmstatus

7.1. Programmstatuswörter im common SNOBOL

Im STRING-System existiert ein common SNOBOL, desser Variablen den Programmstatus beschreiben. Die Variablennamen können bei einer common-Zone freigewählt werden, es entscheidet die Reihenfolge. Die Variablen sind der Reihe nach:

(1) boolean ANCHORV
- true = Anchored -mode
- false = Unanchored-mode

(2) boolean SUCCV
- true = match erfolgreich
- false = match nicht erfolgreich bzw. Prozedur durch Fehlerausgang verlassen

(3) boolean TRACEV
- true = TRACEEIN
- false = sonst

(4) integer MAT1V — Nr. des letzten Zeichens, das vor Beginn des Teilstrings liegt, der "matcht".

(5) integer MAT2V — Nr. des letzten Zeichens, des "matchenden" Teilstrings im String.

(6) integer REMV — Anzahl der Zeichen, die im String nach dem "matchenden" Teilstring noch folgen

(7) boolean TRIMV	true , falls bei EIN auf die Eingabe automatisch TRIM angewandt wird, false sonst. Siehe Prozedur TRIM.
(8) boolean FTRACEV	true, falls FTRACE eingeschaltet, false sonst.
(9) boolean DUMPV	true, wenn Dump eingeschaltet ist, false sonst.

7.2. Prozeduren zur Änderung des Programmstatus

7.2.1. ANCHOR

ANCHOR(0) setzt den "matching"-Algorithmus auf den Mode "unanchored".

ANCHOR(1) setzt Mode auf "anchored".

7.2.2.

TRACE(0) schaltet Trace ab.

TRACE(1) schaltet Trace ein (siehe 6.1.).

7.2.3. FTRACE

```
procedure FTRACE(A,N);
string A; integer N;
```

Für die durch den String A gekennzeichnete Funktion wird der Trace bei N=1 als Prozedureingang, bei N=0 als Prozedurausgang ausgedruckt.

FTRACE(1) schaltet den Funktionstrace ein.

FTRACE(0) schaltet den Funktionstrace aus.

7.2.4. TRIM

TRIM(1) Bei einer Eingabe werden Blanks am Anfang und Ende abgeschnitten.

TRIM(0) Eingabe ohne Veränderung.

7.3. Anfangsprozedur SNOBOL

Mit Aufruf der parameterlosen Prozedur SNOBOL wird der Programmstatus normiert bzw. voreingestellt. Der common SNOBOL hat dann die Werte:

```
ANCHORV  :=  false;
SUCCV    :=  false;
TRACEV   :=  false;
MAT1     :=  0;
MAT2     :=  0;
REMV     :=  0;
TRIMV    :=  false;
FTRACEV  :=  false;
DUMPV    :=  true;
```

7.4. Programmverzweigungen: SUCC und FAIL

Ein Durchlauf des "matching"-Algorithmus wird erfolgreich genannt, wenn ein Teilstring gefunden wird, der

alle Bedingungen erfüllt. Dann wird die Variable SUCCV auf true gesetzt. Ist dies nicht der Fall, wird sie auf false gesetzt. Die Prozeduren SUCC und FAIL erlauben nun eine Verzweigung in Abhängigkeit des Wertes von SUCCV.

```
boolean procedure SUCC;
liefert true falls SUCCV = true;

boolean procedure FAIL;
liefert true falls SUCCV = false;
```

Beispiel:

Die Prozedur DIFFER könnte wie folgt geschrieben werden:

```
boolean procedure DIFFER (A,B);
begin MAT(A, POS(0), B, RPOS(0));
DIFFER := FAIL;
end;
```

8. Programmbeispiele

Beispiel 1: Es sollen alle Teilstrings des Strings 'ELEGANTERE LEUTE', die zwischen zwei 'E' stehen, nach der Länge sortiert ausgedruckt werden.

```
MO STDHP WURDE ERSETZT
MONTAGEOBJEKT  STDHP  WURDE  ERZEUGT
ANFANG PROTOKOLL
000020   'BEGIN'
000030   'PROCEDURE'MAT (X);'CODE';
000040    'INTEGER''PROCEDURE'SV(X);'CODE';
000050   'PROCEDURE'ASS(X);'CODE';
000060   'BOOLEAN''PROCEDURE'SUCC;'CODE';
000070   'BOOLEAN''PROCEDURE'FAIL    ;'CODE';
000080
000090   'ARRAY'A[0:20];
000100   'ARRAY'B,C,D[0:30];
000110   'INTEGER'I,J,K,L;
000120   'REAL'X.Y.Z;
000130    '
000140   ASS(C,'('ELEGANTERE LEUTE')');
000150   PRINT(C);
000160   PRINT('('')');
000170    'FOR'I:=1'STEP'1'UNTIL'16'DO'
000180    'BEGIN'
000190    ASS(A,C);
000200    L1: MAT (A,0,SV(D),'('E')',I,SV(B),'('E')');
000210    'IF'FAIL    'THEN''GOTO'L2;
000220    ASS(A,D,'('E')','('')');
000230
000240    PRINT(B);'GOTO'L1;
000250   L2: PRINT('('---- ')');TYPE(I);
000260   ASS(A,C);
000270   'END';
000280    'END'
ENDE PROTOKOLL

ENDE PS&ALGOLCOMP (28.05)  0.47
GIB KOMMANDOST:TRIBANM.,BOGOL*MO.*STARTEN.

ENDE PS&BIBTRANS (9.06)  0.12
OPERATOR WURDE ERSETZT

ENDE PS&MONTSEGM (13.14)  4.38
```

```
START STDHP

ELEGANTERE LEUTE

L
R
---- 1
 L
UT
---- 2
---- 3
GANT
RE L
---- 4
 LEUT
---- 5
LEGANT
GANTER
---- 6
RE LEUT
---- 7
LEGANTER
---- 8
GANTERE L
---- 9
---- 10
LEGANTERE L
---- 11
GANTERE LEUT
---- 12
---- 13
LEGANTERE LEUT
---- 14
---- 15
---- 16

ENDE STDHP  0.54
```

Beispiel 2: Für die Längen I := 1 bis 4 sollen alle Teilstrings, die zwischen zwei Vokalen stehen, ausgedruckt werden.

```
GIB KOMMANDOSM:MZUF.,SNORGL,910-1180M.
MO STDHP WURDE ERSETZT
MONTAGEOBJEKT  STDHP  WURDE  ERZEUGT
ANFANG PROTOKOLL
000910   'BEGIN'
000920   'PROCEDURE'MAT(X);'CODE';
000930   'PROCEDURE'ASS(X);'CODE';
000940   'BOOLEAN''PROCEDURE'SUCC;'CODE';
000950   'BOOLEAN''PROCEDURE'FAIL    ;'CODE';
000960    'INTEGER''PROCEDURE'SV(X);'CODE';
000970    'INTEGER''PROCEDURE'ANY(X);'CODE';
000975
000980   'ARRAY'A[0:20];
000990   'ARRAY'B,C,D[0:30];
001000    'ARRAY'E[0:20];
001010   'INTEGER'I,J,K,L;
001020   'REAL'X,Y,Z;
001025
001030   ASS(C,'('ELEGANTERE LEUTE')');
001040    ASS(E,'('AEIOU')');
001050   PRINT(C);
001060   PRINT('('')');
001070    'FOR'I:=1'STEP'1'UNTIL'4 'DO'
001080    'BEGIN'
001090    ASS(A,C);
001100    L1: MAT(A,0,SV(D),ANY(E),1,SV(B),ANY(E));
001110    'IF'FAIL(1)'THEN''GOTO'L2;
001120    ASS(A,D,1,'('')');
001130
001140    PRINT(B);'GOTO'L1;
001150   L2: PRINT('('---- ')');TYPE(I);
001160   ASS(A,C);
001170   'END';
001180    'END'
ENDE PROTOKOLL

ENDE PS&ALGOLCOMP (28.05)   0.50
GIB KOMMANDOSM:
```

```
ENDE PS&MONTSEGM (13.14)  4.14

START STDHP

ELEGANTERE LEUTE

L
G
R
T
---- 1
NT
 L
UT
---- 2
LEG
 LE
---- 3
GANT
NTER
RE L
---- 4

ENDE STDHP  0.41
```

Diese beiden Beispiele zeigen die Kombination der SNOBOL-Möglichkeiten mit denen der ALGOL-Syntax (Schleifen etc.).

Beispiel 3: Aus einem eingegebenen Text sollen alle Namen herausgesucht und ausgedruckt werden.

Ein Name sei gebildet nach der Regel

```
Name  ::=  Bu     Name   Bu     Name   Zi
Bu    ::= A ... Z
Zi    ::= 0 ... 9
```

Zunächst wird der String der Namen gebildet und ausgedruckt, dann dieser in die einzelnen Namen aufgespalten.

```
TKKOP.,SNOBOL,*3T.
   005010  'BEGIN'
   005011  'PROCEDURE'ASS(X);'CODE';
   005012  'PROCEDURE'MAT(X);'CODE';
   005013  'INTEGER''PROCEDURE'SV(X);'CODE';
   005014  'INTEGER''PROCEDURE'SPAN(X);'CODE';
   005015  'INTEGER''PROCEDURE'ANY(X);'CODE';
   005016   'BOOLEAN''PROCEDURE'FAIL;'CODE';
   005017  'BOOLEAN''PROCEDURE'SUCC;'CODE';
   005018  'PROCEDURE'EIN(X);'CODE';
   005019  'PROCEDURE'CAS(X);'CODE';
   005020   'INTEGER''PROCEDURE'CAT(X);'CODE';
   005021  'PROCEDURE'SNOBOL;'CODE';
   005022
   005023  'ARRAY'ID,W,BU,BZ,B,C[0:30];
   005030
   005040
   005050   SNOBOL;
   005060
   005070  ASS(ID,'(''')');
   005080  ASS(BU,'('ABCDEFGHIJKLMNOPQRSTUVWXYZ')');
   005090  ASS(BZ,CAT(BU,'('0123456789')'));
   005100  AA: EIN(W);
   005110  'IF'FAIL'THEN''GOTO'CC;
   005120  BB: ASS(W,ANY(BU),SV(B),SPAN(BZ),SV(C),'(''')');
   005125   PRINT(W);PRINT(B);PRINT(C);
   005130  'IF'SUCC'THEN'
   005140  'BEGIN'
   005150  MAT(ID,B,C);
   005160  'IF'FAIL'THEN'ASS(ID,CAT(ID,B,C,'(' ')'));
   005170   'GOTO'BB;
   005175
   005180  'END';
   005185   'GOTO'AA;
   005190  CC: PRINT(ID);
   005200  DD: ASS(ID,0,SV(B),'(' ')','(''')');
   005210  'IF'SUCC'THEN'
   005220  'BEGIN'PRINT(B);'GOTO'DD;
   005230  'END';
   005240  'END'

ENDE TKOPIERE (3.00)  0.11
```

Zeile 5125 enthält einen Testausdruck zur Verfolgung des Programmlaufes.

```
Ü:ÜBIBANM.,KOGOLÜ.

ENDE PS&BIBTRANS (9.06)  0.15
GIB KOMMANDOSÜ:ÜMO.ÜSTARTEÜ.:

ENDE PS&MONTSEGM (13.14)  5.05

START STDHP  Ü:(A+B*C)Ü.
(+B*C)
A

(+*C)
B

(+*)
C

(+*)
C

A + C
A
B
C
```

Ohne Testausdruck soll ein größeres Beispiel gebracht werden.

```
¤STARTE¤.

START STDHP  ¤:
] GRA A B; ! J K C D E;
 D := SIZE A; E := A;
'FOR' I := ! D'DO'
'FOR' J := ! D'DO'
'BEGIN'
 C := NIL;
'FOR' K := ! D'DO'
 C := C ,,( A[ I , K ] CAT B[ K , J ]);¤.

GRA B I J K C D E SIZE FOR DO BEGIN NIL CAT
GRA
B
I
J
K
C
D
E
SIZE
FOR
DO
BEGIN
NIL
CAT

ENDE STDHP  3.43
```

Literaturverzeichnis

1 TR 440 ALGOL 60 - Sprachbeschreibung
TR 440 Unterlagensammlung N31.D1.04

2 Backus, J.W. et.al.:
Revised Report on the Algorithmic Language Algol 60
Num. Math. 4(1963), p. 420
Comm. ACM 6(1963), pp. 1-17

3 Forber, D.J., R.E. Griswold and F.P. Polonsky:
SNOBOL, A String Manipulation Language
JACM, 11(1964), p. 21-30

4 Griswold, R.E., J.F. Poage, I.P. Polansky:
The SNOBOL 4 Programming Language
Prentice-Hall, Englewood Cliffs, New Jersey, 1970

5 Rosendahl, M.:
BOGOL-TAS, ein Weg zur systemnahen Programmierung in ALGOL am TR 440 .
Arbeitsberichte des Rechenzentrums der Ruhr-Universität Bochum Nr. 7206, 1972 .

6 McCarthy, J. et.al.
LISP 1.5 Programmer's Manual
MIT Press, Cambridge 1965

7 Mohn, K.H., Rosendahl, M., H. Zoller:
AIDA, ein Dialogsystem und seine Implementierung in Algol.
1. Fachtagung über Programmiersprachen, München 1971
Berichte der GI Nr. 3, pp. 97-113

8 Jäger, M., Rosendahl, M., Staake, R.:
Einführung in die Listenverarbeitung anhand der Dialogsprache AIDA.
Arbeitsberichte des Rechenzentrums der Ruhr-Universität Bochum Nr. 7203, 1972.

Bisher erschienene Arbeitsberichte des Rechenzentrums der Ruhr-Universität Bochum :

Nr. 7101 : K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, eine Dialogsprache für den TR 440

Nr. 7102 : K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, ein Dialogsystem und seine Implementierung in ALGOL

Nr. 7103 : K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, Manual für den Benutzer

Nr. 7104 : 4. Jahresbericht des Rechenzentrums (Juni 1970 bis Juni 1971)

Nr. 7105 : H. Wupper
WR MBO2 – Ein einfaches Band-Betriebssystem für einen mittleren Rechner

Nr. 7201 : H. Windauer
Existenzsätze zur (0,1,...,R-2,R) - Interpolation

Nr. 7202 : W. Schelongowski
DIATRACE, Ein System zur interaktiven Assemblerprogrammierung

Nr. 7203 : M. Jäger, M. Rosendahl, R. Staake
Einführung in die Listenverarbeitung anhand der Dialogsprache AIDA

Nr. 7204 : R. Mannshardt, P. Pottinger
Einführung in die Benutzung des Teilnehmer-Rechensystems TR 440 in der RUB

Nr. 7205 : 5. Jahresbericht des Rechenzentrums (1.7.1971 bis 30.6.1972)

Nr. 7206 : M. Rosendahl
BOGOL-TAS, ein Weg zur systemnahen Programmierung in ALGOL am TR 440

Nr. 7207 : W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungswechsels von Verbrennungskraftmaschinen (Modulbeschreibung und Eingabekonventionen)

Nr. 7208 : W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungswechsels von Verbrennungskraftmaschinen (Regelmechanismus und Berechnung der Rohrströmung)

Nr. 7209 : H. Ehlich
Anregungen und Kritik zum Betriebs- und Programmiersystem der TR 440

Nr. 7210 : M. Rosendahl
BOGOL-STRING, eine flexible Zeichenkettenverarbeitung in ALGOL 60

Nr. 7211 : H. Camici, H. Claus, H. Ehlich, D. Kipp
Arbeitsbericht über ein Programm zur Haushaltsführung