INSTITUT FÜR
INFORMATIK

Mitteilung 36

# HASY - ein Programm zur syntaktischen Analyse natürlicher Sprachen

M. Mittelstein, B. Nebel, B. Pretschner
P. Schefe

IfI-HH-M-36/76


D 2000 HAMBURG 13
SCHLÜTERSTRASSE 70

I N H A L T

## V o r w o r t

Diese Beschreibung betrifft die Benutzung und die Implementation eines Programms zum Modellieren von Konstituentenstruktur- und Dependenzgrammatiken (HASY = Hamburger Syntax Programm).

Der Neuimplementation in PASCAL, die von M. Mittelstein, B. Nebel und B. Pretschner durchgeführt wurde, liegt ein älteres ALGOL-Programm (s. Mitteilung Nr. 11) von P. Schefe zugrunde. Eine erneute Programmierung war aus verschiedenen Gründen vorteilhaft. Die Erfahrungen mit der älteren Version deckten verschiedene Mängel in der Benutzbarkeit auf. Die neue Fassung bringt hier erhebliche Verbesserungen: dem Benutzer steht jetzt ein überschaubares Inventar von einfachen Kommandos zum Eingeben, Ausgeben und Ändern sowie zum Austesten (Generieren, Analysieren) der Grammatik zur Verfügung. Der Umfang der Grammatik ist praktisch nur noch durch den Benutzer selbst beschränkt.

Die Programmiersprache PASCAL wurde einmal wegen ihrer dynamischen Datenstrukturen sowie ihrer ein systematisches Programmieren unterstützenden Kontrollelemente verwendet, zum andern verspricht eine Erweiterung des Programms, wie sie in der ALGOL-Version z.T. schon vorliegt, einen größeren Anwendungsbereich innerhalb anderer Projekte, die ebenfalls PASCAL benutzen. Bei einem Ausbau der semantischen Analyse ist hier vor allem an Programme zur Kommunikation mit Datenbanken und szenenanalysierenden Systemen zu denken.

## Zu den linguistischen Grundlagen

### Konstituentenstrukturgrammatik

Die Methode der Zerlegung von Sätzen in ihre unmittelbaren Bestandteile (Konstituenten) war eine wichtige Voraussetzung für die Formalisierung der Beschreibung natürlicher Sprachen. Trotz vielfacher methodischer und inhaltlicher Kritik behält das Modell in bestimmten Grenzen seine Gültigkeit. Sätze werden mittels bestimmter syntaktischer Tests binär und hierarchisch bis zu Morphemen oder Wörtern gegliedert. Die Formalisierung ist in der

sogenannten Chomsky-normalen kontextfreien Phrasenstrukturgrammatik (NCFG) gegeben. Eine NCFG ist ein formales System

$$G(V_N, V_T, S, P)$$

wobei $V_N$, $V_T$ endliche, nicht-leere Mengen mit $V_N \cap V_T = \{\}$ ,
S ein ausgezeichnetes Symbol, $S \in V_N$,
P die Menge der Produktionsregeln der Form ('→' lies 'wird ersetzt durch')

(a) $A \rightarrow B + C$ mit $A,B,C \in V_N$, (binäre Regeln)

(b) $A \rightarrow a$ mit $A \in V_N$ und $a \in V_T$ (terminierende Regeln)

z.B. S → NP + VP, N → Glas.
$V_N$ wird auch Menge der syntaktischen Kategorien oder Nonterminals genannt.
Die von G erzeugte Sprache L(G) ist die Menge aller mittels der Produktionsregeln P aus S ableitbaren Ketten von Elementen aus $V_T$ · G liefert zu jedem Element aus L(G) eine Strukturbeschreibung, darstellbar als Ableitungsbaum oder indizierter Klammerausdruck. (Näheres s. Chomsky 1963)

Die Binarität der Phrasenstrukturregeln könnte man ausschließlich technisch-formal rechtfertigen. Eine NCFG läßt sich durch ein Programm leichter behandeln. Es genügt der Hinweis auf die Tatsache, daß sich zu jeder CFG eine NCFG schwach äquivalent konstruieren läßt, eine CFG also, die dieselbe Sprache erzeugt.

Beispiel:

$G (V_N, V_T, S, P)$ mit

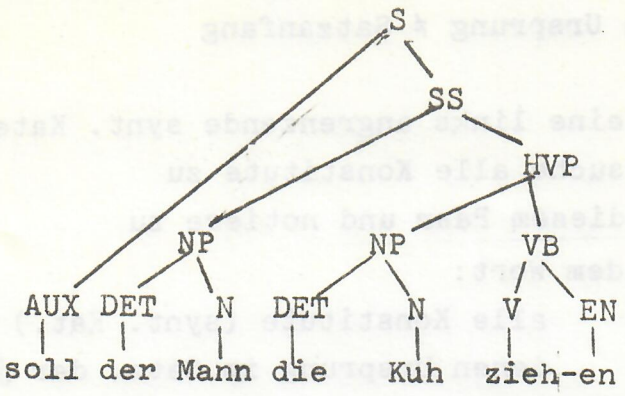
$V_N = \{S, SS, NP, AVP, HVP, N, DET, AUX, HV, VB, EN\}$

$V_T$ ergibt sich aus den Regeln

P = { S → AUX + SS, N → Kuh Wagen Mann,
S → NP + AVP, DET → der die den,
SS → NP + HVP, V → zieh schieb,
NP → DET + N, EN → en,
AVP → AUX + HVP, AUX → soll kann }
HVP → NP + VB,
VB → V + EN,

(Dabei wird für die terminierenden Regeln - Typ (b) - eine abkürzende Schreibweise verwendet)

erzeugt z.B. den Satz mit der Strukturbeschreibung:




Eine formale Grammatik ist die Voraussetzung

1. für ein Verfahren zur Aufzählung aller korrekten Sätze einer Sprache,
2. für ein Verfahren zur Entscheidung darüber, ob ein gegebener Satz zu der durch die Grammatik definierten Sprache gehört.

Zu 1. liefert das Programm kein systematisches Verfahren. Sätze können per Zufall oder durch Steuerung des Benutzers im 'Generiermodus' erzeugt werden.

Zu 2. sind verschiedene Verfahren für kontextfreie Sprachen bekannt. Die Entscheidung über die Zugehörigkeit eines Satzes zu der durch die Grammatik definierten Sprache geschieht durch einen 'Parser', der ggfls. eine Strukturbeschreibung für einen Satz liefert. Der hier verwendete Parsingalgorithmus ist eine besondere Form des Cocke-Algorithmus (vgl.Hays 1967)für NCFGn:

Notiere zu jedem Wort (Morphem) der Eingabe alle syntaktischen Kategorien. (z.B. zu 'der': DET, RELPRO)
Vom 2. bis zum letzten Wort führe aus:
Soweit eine bei diesem Wort notierte syntaktische Kategorie zu finden ist, deren Ursprung ≠ Satzanfang
und
soweit jeweils noch eine links angrenzende synt. Kategorie zu finden ist, dann suche alle Konstitute zu diesem Paar und notiere zu dem Wort:
alle Konstitute (synt. Kat.) und deren Ursprung im Satz, der jeweils gleich dem Ursprung der linken Konstituente ist, sowie die Position der jeweiligen Konstituenten (vgl. u. Wegetabelle)
Wenn ein Konstitut 'S' gefunden wird, das den ganzen Satz überdeckt, wird die Eingabe (voll) akzeptiert. ('*' in der Wegetabelle)

Dependenzgrammatik

Es ist relativ einfach, irgendeine Grammatik zur Erzeugung einer bestimmten Satzmenge anzugeben. Schwierig ist die Feststellung, ob eine Grammatik und die durch sie gelieferten Strukturbeschreibungen adäquat sind. Ein wichtiger Adäquatheitstest ist die Prüfung der Möglichkeit, die Struktur semantisch zu interpretieren: ist mit der Bildung eines Konstituts aus zwei Konstituenten auch die Bildung eines Bedeutungskomplexes möglich? Diese Frage ist sicher bei Erweiterung des Programms um eine semantische Komponente am besten zu beantworten (vgl. Schefe 1975). Ein wichtiger Schritt in diese Richtung ist jedoch bereits die Konstruktion von Dependenzstrukturen (vgl. Schefe 1976 ).

Dependenstrukturen und Konstituentenstrukturen hängen eng miteinander zusammen: Es ist nützlich, vor dem Schreiben einer NCFG für einen Sprachausschnitt eine Dependenzanalyse zu betreiben (vgl. Kratzer et al. 1973); anderseits setzt jede Dependenzstruktur eine Konstituentenstruktur voraus.

Das hier verwendete Modell trägt dem Rechnung dadurch, daß die Dependenzstruktur eines Satzes aus seiner Phrasenstruktur ermittelt wird.

Entsprechend läßt sich ein Dependenzsystem bezüglich einer NCFG formal definieren als

$$D(V_P, V_{TR}, PR, F)$$
bezüglich $$G(V_N, V_T, S, P)$$

wobei $V_P \subseteq V_N$, die Menge der präterminalen Elemente aus $V_N$
$V_{TR} \subseteq V_N$ die Menge der 'Translatoren'
$PR \subseteq V_P \times N$, eine Abbildung der präterminalen Kategorien in die Menge der natürlichen Zahlen, die 'Prioritäten'
F eine Abbildung von Phrasenstrukturen auf Dependenzstrukturen mittels 'Prioritäten' und 'Translatoren' (informale Beschreibung s.u.)

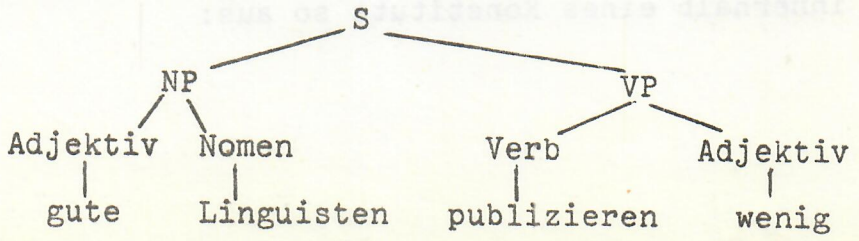
Dependenzbeziehungen werden zunächst durch Prioritäten ausgedrückt. Ein Element A 'regiert' ein Element B, formal A → B, wenn es die niedrigere Priorität besitzt, z.B. ergeben sich aus den Prioritäten für

Verb: 1
Nomen: 2
Adjektiv: 3

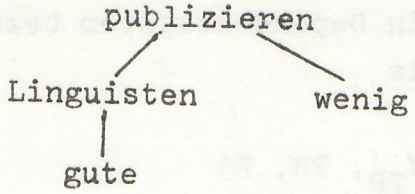
die Dependenzbeziehungen:

Verb → Nomen
Verb → Adjektiv
Nomen → Adjektiv

Dependenzbeziehungen zwischen Wortklassen wurden zunächst von Tesnière (1959) informal postuliert. Diese Beziehungen werden dann für einzelne Sätze durch die Wörter bzw. Morpheme konkretisiert, z.B. aus der Phrasenstruktur:



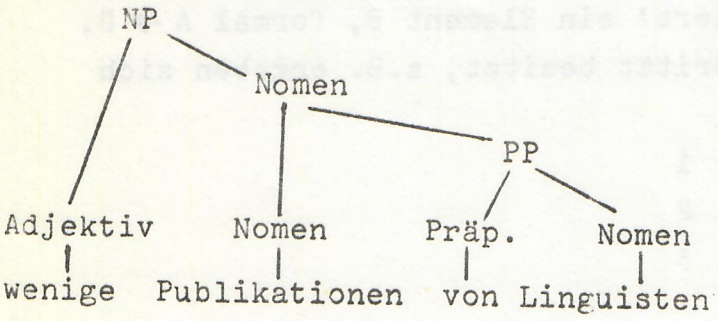
ergibt sich die Dependenzstruktur:



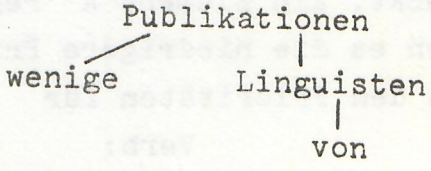

dadurch, daß für jedes Konstitut aufgrund eines Prioritätsvergleiches das regierende Element ermittelt wird, z.B. für NP: Linguisten. Translatoren modifizieren dieses Verfahren für ein 'rankshift' (vgl. Winograd 1972, S. 18). Diese Erscheinung, die Abhängigkeit eines Elements von einem der Priorität nach gleichgeordneten anderen Element, z.B. eines Nomens von einem Nomen, wurde von Tèsniere als Translation bezeichnet.

Beispiel:

Phrasenstruktur




Dependenzstruktur




mit dem Translator PP, der ein Nomen mit 'rankshift' enthält.

Durch die Spezifikation von Translatoren können auch Elemente niedrigerer Priorität von einem Element höherer Priorität abhängen, z.B. ein Verb von einem Nomen in unserem Beispiel.

Im einzelnen sieht die Prozedur zur Ermittlung des Regenten/Dependenten innerhalb eines Konstituts so aus:

Wenn gilt: (Rechte Konstituente ist Translator) oder
(Priorität(Linke Konstituente) <=
Priorität(Rechte Konstituente)) und
nicht (Linke Konstituente ist Translator),
dann ist der Regent der linken Konstituente auch der
Regent des Konstituts,
sonst der der rechten Konstituente.

Weiterer Ausbau des Programms

Die Implementation des Syntax-Parsers ist so angelegt, daß nach den Erfahrungen mit den ALGOL-Implementierungen verschiedene Schnittstellen für die Erweiterung angelegt sind. Die nächsten, in ALGOL z.T. schon realisierten Erweiterungen wären

- eine sinnsemantische Komponente: Formulierung von Bedeutungspostulaten, Begriffshierarchien, Selektionsbeschränkungen als semantisches Netz mit paradigmatischen und syntagmatischen Relationen

- eine Transformationskomponente: Erkennung syntaktischer Paraphrasen (Aktiv - Passiv z.B.)

- eine Frage-Antwort-Komponente mit Frageoperatoren, Beantwortung von Ja/Nein-Fragen und W-Fragen mit Deduktionen (Auffinden impliziter Information)

- eine referenzsemantische Komponente in Form einer relationalen Datenbasis oder eines Simulationsmodells für Szenen (vgl. Winograds 'blocks world')

u.a.

## Die Benutzung von HASY

Die Benutzung von HASY ist nach einer Einführung in die oben angegebenen Formate der Phrasenstruktur- und Dependenzgrammatik fast selbsterklärend, da der Benutzer mit "?" verschiedene helfende Hinweise abrufen kann. Die Erklärung stützt sich hier auch auf diese Texte.

Das Programm wird aufgerufen mit
(Die Benutzereingaben werden jeweils unterstrichen;
'↵' bedeutet die Betätigung der CR-Taste):

```
.RUN HASY ↵
```

Das Programm verlangt dann die Angabe der Namen von Dateien für die Grammatik bzw. erzeugte Sätze und Satzanalysen.
Es muß eine Eingabedatei existieren, die auch leer sein kann ('Syntaxein'). Die Datei für die Ausgabe der Grammatik kann durch die Angabe eines Namens - ebenso wie die Protokolldatei - auch kreiert werden. Beispiel:

```
SYNTAXEIN = ANFANG ↵
SYNTAXAUS = ENDE ↵
PROTOKOLL = PROT ↵

*↵


HAMBURGER SEMANTIK - PARSER, FASSUNG VOM 14-SEP-76


ES WURDEN     0.000 SEKUNDEN ZUM EINLESEN BENOETIGT
```

Danach geht das Programm automatisch in den Kommandomodus, angezeigt durch:

```
EKOMMANDO ]*?↵


EKOMMANDO  ]*
?                "."     BEENDIGUNG DES PROGRAMMLAUFES
?
?                "A"     UEBERGANG IN DEN ANALYSE MODUS
?                "E"     UEBERGANG IN DEN ERWEITERUNGS MODUS
?                "G"     UEBERGANG IN DEN GENERIER MODUS
?                "L"     AUSGABE DER GESPEICHERTEN DATEN AUF DIE
?                        OBEN ALS SYNTAXAUS BENANNTE DATEI
?
?                "?"     DIESE NACHRICHT
```

Durch Eingabe des "?" hat der Benutzer hier anschließend die erklärenden Hinweise für den Kommandomodus abgerufen.

In der Annahme, daß die Eingabedatei noch leer ist, wird man zunächst in den Erweiterungsmodus übergehen (Befehl "E"):

```
[KOMMANDO ]*E↵

[ERWEITERN]*?↵

[ERWEITERN]*
?       "."     BEENDIGUNG DES ERWEITERUNGS MODUS UND RUECKEHR IN
?               DEN KOMMANDO MODUS
?
?               KOMMANDOFORMATE
?
?       "XN <NON> <ZAHL>"                       FUER NONTERMINALS
?       "XR <NON> <-- <NON> + <NON>"            FUER REGELN
?       "XL <NON> <-- <TERM>"                   FUER LEXIKONEINTRAEGE
?       "XT <TERM>"                             FUER TRANSLATOREN
?       "S <NON> <ZAHL>"                        FUER DAS NEUE SETZEN DER
?                                               STUFEN VON NONTERMINALS
?
?               WOBEI "X" EINER DER FOLGENDEN BUCHSTABEN SEIN DARF
?
?       "E"     FUER EINSETZEN
?       "S"     FUER SUCHEN
?       "L"     FUER LOESCHEN
?
?       <NON>   IST EIN NONTERMINALER BEGRIFF
?       <TERM>  IST EIN TERMINALER BEGRIFF
?       <ZAHL>  IST EINE BELIEBIGE GANZE ZAHL OHNE VORZEICHEN
?
?          ANSTELLE VON <NON> UND <TERM> DARF AUCH EIN "*" STEHEN,
?          WENN DER BEGRIFF NICHT GENAU SPEZIFIZIERT WERDEN SOLL
?
?       "?"     DIESE NACHRICHT
```

Nachdem sich der Erweiterungsmodus gemeldet hat, erfolgt auf die Eingabe eines "?" eine entsprechende Erklärung der Kommandoformate, die im folgenden noch durch Beispiele konkretisiert werden. Zunächst muß der Benutzer gemäß der Definition der Phrasenstrukturgrammatik die Menge der syntaktischen Kategorien (wenigstens ein Nonterminal) eingeben. Erst dann kann er aus den eingegebenen Symbolen binäre Regeln und terminierende Regeln (Lexikoneinträge) bilden.

Kommandos für die Bearbeitung der Menge der Nonterminals bestehen aus vier Teilen:

der erste Buchstabe spezifiziert, ob ein Nonterminal eingesetzt ("E"), auf Vorhandensein/Nichtvorhandensein geprüft ("S") oder gelöscht ("L") werden soll;
der zweite Buchstabe ist "N";
dann folgt das Nonterminal selbst
und die Prioritätsstufe ( ZAHL )
(wenn sie weggelassen wird, ist ihr Wert 0)

Beispiele:

```
[ERWEITERN]*EN S↵
S                0   EINSETZEN  (J/N) :↵

[ERWEITERN]*EN NP↵
NP               0   EINSETZEN  (J/N) :↵

[ERWEITERN]*EN VP↵
VP               0   EINSETZEN  (J/N) :↵

[ERWEITERN]*EN N 2↵
N                2   EINSETZEN  (J/N) :↵

[ERWEITERN]*EN DET 3↵
DET              3   EINSETZEN  (J/N) :↵

[ERWEITERN]*EN AJ 3↵
AJ               3   EINSETZEN  (J/N) :↵

[ERWEITERN]*EN V 1↵
V                1   EINSETZEN  (J/N) :↵

[ERWEITERN]*EN ANP 0↵
ANP              0   EINSETZEN  (J/N) :↵

[ERWEITERN]*SN *↵
*              SUCHEN     (J/N) :↵

S                0
NP               0
N                2
DET              3
AJ               3
ANP              0
VP               0
V                1

[ERWEITERN]*ET ANP↵
```

Im letzten Kommando wurde die vorher definierte Kategorie "ANP" als Translator deklariert.

Alle andern Kommandos des Erweiterungsmodus laufen nach demselben Grundmuster ab:

```
[ERWEITERN]*ER S NP VP↵
S          <-- NP          + VP          EINSETZEN  (J/N) :↵

[ERWEITERN]*EL N LINGUISTEN↵
N          <-- LINGUISTEN                EINSETZEN  (J/N) :↵
```

Die eingegebene Grammatik kann man sich durch Suchkommandos ausgeben lassen, z.B.:

```
[ERWEITERN]*SL * *↵
*          <-- *                         SUCHEN     (J/N) :↵

AJ         <-- VIEL
AJ         <-- WENIG
N          <-- ASSISTENTEN
N          <-- INFORMATIKER
N          <-- LINGUISTEN
PRAE       <-- VON
V          <-- PROGRAMMIEREN
V          <-- SCHREIBEN

[ERWEITERN]*SR * * *↵
*          <-- *          + *            SUCHEN     (J/N) :↵

ANP        <-- PRAE       + N
N          <-- N          + ANP
NP         <-- DET        + N
S          <-- NP         + VP
VP         <-- V          + AJ

[ERWEITERN]*EL  DET ALLE↵
DET        <-- ALLE                      EINSETZEN  (J/N) :↵

[ERWEITERN]*.↵
```

Zum Testen der Grammatik geht man zweckmäßigerweise zunächst in den Generiermodus über:

```
KOMMANDO ]*G

GENERIEREN]*?


GENERIEREN]*
?                    "."       BEENDIGUNG DES GENERIER MODUS UND UEBERGANG IN
?                              DEN KOMMANDO MODUS
?
?                    DURCH GEBEN EINES <CR> WIRD DIE GENERIERUNG EINES SATZES
                     ENTSPRECHEND DEN GESETZTEN SCHALTERN AUSGEFUEHRT. DIE
                     VOREINSTELLUNG IST 'ZUFALL', 'REKURSIONSGRAD 4' UND TERMI-
                     NATIONSGRAD 4'. DURCH EINGABE EINES "-" VOR DEN BUCHSTABEN
                     (BEI "Z","D","O" UND "A") KOENNEN DIE SCHALTER ZURUECKGESETZT
                     WERDEN. DIE SCHALTER GELTEN BIS AUF WIDERRUF

                     "Z"       DIE GENERIERUNG ERFOLGT PER ZUFALL
                     "D"       ES WERDEN DIE BENUTZTEN REGELN UND ZWISCHEN-
                               STUFEN DES SATZES AUSGEDRUCKT
                     "O"       VOR- UND NACHSILBEN WERDEN BEI DER AUSGABE AN
                               DIE ENTSPRECHENDEN WOERTER ANGEHAENGT
                     "A"       DIE GESAMTE AUSGABE ERFOLGT AUF DEN DRUCKER;
                               DER GENERIERTE SATZ WIRD ZUSAETZLICH AUF DER
                               BENUTZERSTATION AUSGEGEBEN

                     "R#"      REKURSIONSWAHRSCHEINLICHKEIT, WOBEI "#" EINE
                               ZAHL ZWISCHEN "1" UND "9" SEIN MUSS
                     "T#"      TERMINATIONSWAHRSCHEINLICHKEIT, WOBEI "#" EINE
                               ZAHL ZWISCHEN "1" UND "9" SEIN MUSS

                     "="       AUSGABE DER GESETZTEN SCHALTER

                     "?"       DIESE NACHRICHT


GENERIEREN]*.

KOMMANDO ]*.

EXIT
1.26    ]
```

Die Generierung eines Satzes erfolgt gemäß seiner syntaktischen Struktur von "oben nach unten" und "von links nach rechts"; sie beginnt mit der Auswahl einer Regel, die das Symbol "S" auf der linken Seite hat, in unserem Beispiel als "S → NP + VP".
Dann wird "NP" expandiert, bis - falls dies möglich ist - alle Nonterminals durch Terminals ersetzt worden sind.

Für den Fall, daß dieser Prozeß zufällig verläuft, sind die Parameter 'Rekursionswahrscheinlichkeit' und 'Terminationswahrscheinlichkeit' wichtig. Wenn für ein Nonterminal sowohl binäre als auch terminierende Regeln existieren, wird durch die Angabe der Zahl 3

z.B. eine terminierende Regel nur in 3 von 10 Fällen gewählt. Entsprechend wird entschieden, wenn unter den binären Regeln, die dasselbe Nonterminal auf der linken Seite haben, rekursive und nicht-rekursive zu finden sind.

Zunächst einige Beispiele für zufallsgenerierte Sätze, dann die Erzeugung eines Satzes, die durch den Benutzer gesteuert wird:

---

```
[GENERIERE]*↵
ALLE ASSISTENTEN SCHREIBEN WENIG
[GENERIERE]*↵
ALLE INFORMATIKER VON INFORMATIKER SCHREIBEN WENIG
[GENERIERE]*↵
ALLE ASSISTENTEN VON ASSISTENTEN SCHREIBEN WENIG
```

---

```
[GENERIERE]*-Z↵
   1.  S            <-- NP          + VP
WELCHE REGEL?:1↵
   1.  NP           <-- DET         + N
WELCHE REGEL?:1↵
   1.  DET          <-- ALLE
WELCHE REGEL?:1↵

DET          <-- ALLE
ALLE
   1.  N            <-- N           + ANP
   2.  N            <-- ASSISTENTEN
   3.  N            <-- INFORMATIKER
   4.  N            <-- LINGUISTEN
WELCHE REGEL?:1↵
   1.  N            <-- N           + ANP
   2.  N            <-- ASSISTENTEN
   3.  N            <-- INFORMATIKER
   4.  N            <-- LINGUISTEN
WELCHE REGEL?:2↵
```

```
N           <-- ASSISTENTEN

ALLE ASSISTENTEN
   1.  ANP          <-- PRAE          + N

WELCHE REGEL?:4↵

WELCHE REGEL?:1↵
   1.  PRAE         <-- VON

WELCHE REGEL?:1↵


PRAE        <-- VON

ALLE ASSISTENTEN VON
   1.  N            <-- N             + ANP
   2.  N            <-- ASSISTENTEN
   3.  N            <-- INFORMATIKER
   4.  N            <-- LINGUISTEN

WELCHE REGEL?:4↵


N           <-- LINGUISTEN

ALLE ASSISTENTEN VON LINGUISTEN
   1.  VP           <-- V             + AJ

WELCHE REGEL?:1↵

   1.  V            <-- SCHREIBEN
   2.  V            <-- PROGRAMMIEREN

WELCHE REGEL?:1↵


V           <-- SCHREIBEN

ALLE ASSISTENTEN VON LINGUISTEN SCHREIBEN
   1.  AJ           <-- WENIG
   2.  AJ           <-- VIEL

WELCHE REGEL?:2↵


AJ          <-- VIEL

ALLE ASSISTENTEN VON LINGUISTEN SCHREIBEN VIEL
```

Der Analysemodus dient vor allem zur Überprüfung der Mehrdeutigkeit der Grammatik und zur Untersuchung der Strukturbeschreibungen:

```
[KOMMANDO ]*A

[ANALYSE  ]*?


[ANALYSE    ]*
?                   NACH DEM ERSCHEINEN DES STERNCHENS KANN EIN SATZ
?                   ZUR ANALYSE EINGEGEBEN WERDEN
?
?                   "."       BEENDIGUNG DES ANALYSE MODUS UND UEBERGANG
?                             IN DEN KOMMANDO MODUS
?
?                   IM ANALYSE MODUS KOENNEN DIE FOLGENDEN SCHALTER GESETZT
?                   WERDEN. DAS ZURUECKSETZEN DER SCHALTER ERFOLGT DURCH
?                   ZUSAETZLICHE EINGABE EINES "-" VOR DEM BUCHSTABEN.
?                   WENN MEHRERE SCHALTER GLEICHZEITIG GESETZT ODER ZURUECK-
?                   GESETZT WERDEN SOLLEN, SO IST NUR AN DEN ANFANG DER
?                   ZEICHENFOLGE EIN "!" ZU SETZEN. DIE GESETZTEN SCHALTER
?                   GELTEN BIS AUF WIDERRUF.
?
?                   "!T"      TESTLAUF
?                   "!W"      AUSGABE DER WEGETABELLE
?                   "!K"      AUSGABE DER KONSTITUENTENSTRUKTUREN
?                   "!D"      AUSGABE DER DEPENDENZSTRUKTUREN
?                   "!A"      DIE GESAMTE AUSGABE ERFOLGT AUF DEN DRUCKER
?
?                   "="       AUSGABE DER GESETZTEN SCHALTER AUF DAS TERMINAL
?
?                   "?"       DIESE NACHRICHT
```

Der folgende Dialog zeigt die Analyse eines Satzes mit Wegetabelle, Konstituentenstrukturen und Dependenzstrukturen:

```
[ANALYSE  ]*ALLE INFORMATIKER PROGRAMMIEREN WENIG↵

SATZ IST  1-DEUTIG

ALLE INFORMATIKER PROGRAMMIEREN WENIG

[ANALYSE  ]*!TWKD↵

ALLE INFORMATIKER PROGRAMMIEREN WENIG
```

| | NONTERMINAL | STUFE | REGEL | REGIERENDES MERKMAL |
|---|---|---|---|---|
| 1 | DET | 3 | | ALLE |
| 2 | N | 2 | | INFORMATIKER |
| 3 | NP | 2 | ( 1) DET + ( 2) N | INFORMATIKER |
| 4 | V | 1 | | PROGRAMMIEREN |
| 5 | AJ | 3 | | WENIG |
| 6 | VP | 1 | ( 4) V + ( 5) AJ | PROGRAMMIEREN |
| 7* | S | 1 | ( 3) NP + ( 6) VP | PROGRAMMIEREN |

KONSTITUENTENSTRUKTUREN ZU SATZ 7

```
S (PROGRAMMIEREN)
    NP (INFORMATIKER)
        DET (ALLE)
            ALLE
        N (INFORMATIKER)
            INFORMATIKER
    VP (PROGRAMMIEREN)
        V (PROGRAMMIEREN)
            PROGRAMMIEREN
        AJ (WENIG)
            WENIG
```

DEPENDENZSTRUKTUREN ZU SATZ 7

```
PROGRAMMIEREN
    WENIG
    INFORMATIKER
        ALLE
```

Die Wegetabelle listet alle Konstitute auf, die während des Parsings gebildet werden. Jede durch "==...=" gekennzeichnete Untertabelle enthält alle Konstitute, die an dieser Stelle des Satzes, bezeichnet durch die Position des i-ten Wortes, enden. Zugleich verzeichnet sie deren Prioritätsstufe, die beiden Konstituenten mit Verweisen auf die entsprechende Untertabelle sowie den jeweiligen Regenten, die in den Konstituentenstrukturen dem Konstitut jeweils in runden Klammern hinzugefügt sind. Die Strukturen sind hier aus programmiertechnischen Gründen gegenüber der üblichen Darstellung um die Diagonale gespiegelt.

Will man die bearbeitete Grammatik abspeichern, so gibt man "L" im Kommandomodus ein; nach Beendigung des Programmlaufs kann man sich die Grammatik noch einmal ausdrucken lassen:

```
[ANALYSE   ]*. ↵

[KOMMANDO ]*L ↵

[KOMMANDO ]*. ↵

EXIT
[5.86    ]

.TYPE ENDE ↵
[15:13:05]
NONTERMINALS    ==========================================

S                   0
NP                  0
N                   2
DET                 3
AJ                  3
ANP                 0
PRAE                4
VP                  0
V                   1
.
TRANSLATOREN    ==========================================

ANP
.
```

```
REGELN          ========================================

ANP         <--- PRAE          + N
N           <--- N             + ANP
NP          <--- DET           + N
S           <--- NP            + VP
VP          <--- V             + AJ
.
LEXIKON         ========================================

AJ          <--- VIEL
AJ          <--- WENIG
DET         <--- ALLE
N           <--- ASSISTENTEN
N           <--- INFORMATIKER
N           <--- LINGUISTEN
PRAE        <--- VON
V           <--- PROGRAMMIEREN
V           <--- SCHREIBEN
.
[0.56   ]
```

## Literaturverzeichnis


N. Chomsky, Formal properties of grammars, In: R.D. Luce et al (Eds.), Handbook of mathematical psychology, New York 1963, S. 323-418

D.G. Hays, Dependency Theory, A formalism and some observations, In: Language 40 (1964), S. 511-525

D.G. Hays, Introduction to Computational Linguistics, New York 1967 (Elserier)

D.E. Knuth, Semantics of Context-Free Languages, In: Mathematical Systems Theory 2(1968), S. 127-131

A. Kratzer, E. Pause, A.v. Stechow, Einführung in die Theorie und Anwendung der Generativen Syntax, 1. Halbband: Theorie, Frankfurt 1973

A. Kratzer, E. Pause, A.v. Stechow, Einführung in die Theorie und Anwendung der Generativen Syntax, 2. Halbband: Anwendung, Frankfurt 1974

S. Kuno, Computer Analysis of Natural Languages, In: J.T. Schwartz (Ed.), Mathematical Aspects of Computer Science, Proc. Symp. Appl. Math. 19, Providence 1967, S. 51-110

J. Palme, Making Computers understand natural language, In: N.V. Findler, B. Meltzer (Eds.), Artificial Intelligence and Heuristic Programming, Edinburgh 1971, S. 199-243

R. Rustin (Ed.), Natural Language Processing, New York 1973

P. Schefe, Beschreibung eines Programms für das Konstruieren und Testen von formalen Grammatiken im Dialog mit einer Rechenanlage, Mitteilung Nr. 11, Institut für Informatik, Hamburg 1975

P. Schefe, Die Simulation von Grammatikmodellen und ihr Einsatz im Hochschulunterricht, In: K. Braunmüller, W. Kürschner (Hrsg.), Grammatik, Akten des 10. Ling. Koll. Tübingen 1975, Tübingen 1976, S. 371-382

P. Schefe, Ein dependentiell-transformationeller Parser für den Mensch-Maschinen-Dialog in natürlicher Sprache, In: J.H. Laubsch, H.-J. Schneider, Workshop Dialoge in natürlicher Sprache und Darstellung von Wissen, Stuttgart 1976a, S. 175-191

T. Winograd, Understanding natural language, Edinburgh 1972

L. Tesnière, Éléments de syntaxe structurale, Paris 1959

## 0. Einleitung

### 0.1 Warum entstand dieses Programm

Wie oben schon erwaehnt existiert ein ALGOL - HASE (Hamburger Semantik Parser), der allerdings einige Nachteile hat: ALGOL besitzt keine dynamischen Datenstrukturen, so dass eine optimale Speicherplatzverwaltung nicht moeglich ist; ALGOL - Prozeduren koennen nicht an PASCAL - Programme angebunden werden, was in einem spaeteren Stadium der Programmentwicklung notwendig werden koennte; ausserdem ist der ALGOL - HASE im Laufe meherer Jahre entstanden, und dadurch z.T. unstrukturiert, so dass sich die Notwendigkeit ergab, das Programm neu zu schreiben .

### 0.2 Ueberblick ueber den Stand der Implementation

An sich war geplant, das gesamte ALGOL Programm umzuschreiben. Diese Aufgabe liess sich aber innerhalb von vier Wochen, die wir zur Verfuegung hatten, nicht bewaeltigen. Es wurde lediglich eine Fassung erstellt, die die Syntaxanalyse enthaelt, und einige ueber das ALGOL Programm hinausgehende Teile, wie z.B. eine komfortable Dialogsprache und einen Modus zum interaktiven Aendern der eingelesenen Grammatik. Das wurde besonders im Hinblick darauf implementiert, dass das Programm als Uebungsprogramm fuer Linguistikstudenten verwendet werden soll.

Der Semantik - Teil, der schon im ALGOL - HASE vorhanden ist, und einige Teile, die noch geplant sind, koennen ohne weiteres dem Programm hinzugefuegt werden, weil die Programm- und Datenstrukturen bereits unter Beruecksichtigung dieser Erweiterung konzipiert wurden. Die folgende Dokumentation soll die Implementation dieser Features so weit wie moeglich unterstuetzen.

### 0.3 Ueberblick ueber das Programm

Die jetzige Fassung des Programmes erlaubt es, eine Grammatik von einer Datei einzulesen (siehe Abschnitt 2), diese interaktiv zu aendern (siehe Abschnitt 6), die erzeugte Grammatik auszugeben (siehe Abschnitt 3), Saetze zu analysieren, ihre Dependenz- und Konstituentenstruktur auszugeben (siehe Abschnitt 4) und Saetze generiern zu lassen (siehe Abschnitt 5).

Die Daten, auf denen das Programm im wesentlichen operiert, sind die Nonterminale, Terminale und die syntaktischen Regeln, also die Grammatik. Ihre Darstellung sei im folgenden kurz erlaeutert.

Die Nonterminale sind in einem binaeren Baum, der aus RECORDs des Typs TYPNONTERMINAL besteht, organisiert, so dass das alphabetische Suchen unterstuetzt wird. In jedem dieser RECORDs sind alle wesentlichen Informationen ueber ein Nonterminal zusammengefasst:

Der Bezeichner des Nonterminals [NTNAME],

die Prioritaetsstufe [NTSTUFE]

um einen Translator handelt [NTTRANSLATOR] (diese Information wird fuer die Transformation einer Konstituentenstruktur in eine Dependenzstruktur benoetigt),

wenn es sich um einen Transformationsausloeser handelt, ein Verweis auf die Transformationsregeln [NTTRANSFORMATION],

wenn das Nonterminal rechte Konstituente einer oder mehrerer Regeln ist, ein Verweis auf diese Regeln [NTREGEL],

und je ein Verweis auf den rechten und linken Nachbarn im Baum [NTRECHTER und NTLINKER].

Bei dieser Version des Programmes existieren noch zwei Felder, die das Generieren unterstuetzen, ein Zeiger auf eine Kette von Verweisen auf expandierende binaere Regeln [NTPRODREGEL] und ein Zeiger auf eine Kette von Verweisen auf Terminale, die expandierenden terminierenden Regeln entsprechen [NTPRODTERMINALS] .

Die Terminale [RECORDs vom Typ TYPTERMINAL] sind aehnlich wie die Nonterminale in einem binaeren Baum gespeichert. Da jede Bezeichnung eines Terminals mehrdeutig sein kann, sowohl in semantischer als auch in syntaktischer Hinsicht, zeigt ein Zeiger [TLSEMANTIK] auf eine Kette von von RECORDs vom Typ TYPSEMANTIK, die die moeglichen Beschreibungen enthalten. Zusaetzlich wird noch der Bezeichner [TLNAME] und die Verweise auf den linken und rechten Nachbarn im Baum [TLRECHTER und TLLINKER] im "Terminal" - RECORD abgespeichert.

In den "Semantik" - RECORDs sind in dieser Fassung paradoxerweise im wesentlichen nur syntaktische Informationen gespeichert, naemlich die syntaktische Kategorie, d.h. die terminierende Regel des Terminals [SMKATEGORIE]. Das Feld SMTERMINAL, das ein Rueckverweis auf den terminalen Bezeichner ist, wird lediglich bei der Ausgabe der Dependenzstrukturen benoetigt. SMNAECHSTER erlaubt es die "Semantik" - RECORDs in einer linearen Liste zu organisieren. Die restlichen Felder koennen bei einer spaeteren Fassung des Programmes ( mit Semantik - Teil) dazu benutzt werden, paradigmatische [SMPARADIG] und syntagmatische [SMSYNTAGMATISCH] semantische Beziehungen abzuspeichern. Die Felder SMFRAGEWORT und SMFREIKOMBINIERBAR sind eine zusaetzliche Beschreibung.

Die binaeren Regeln werden durch RECORDs vom Typ TYPREGEL repraesentiert. Die linke und rechte Konstituente sowie das Konstitut einer Regel werden durch ihre Namensvettern, die auf Nonterminale zeigen, im "Regel" - RECORD vertreten. Zusaetzlich existiert ein Feld RGNAECHSTER. Dieses ermoeglicht es, die Regeln in linearen Listen zu organisieren. Es werden alle Regeln mit gleicher rechter Konstituente in solch einer Liste zusammengefasst, auf die dann, wie oben schon erwaehnt, der Zeiger NTREGEL desjenigen "Nonterminal" - RECORDs zeigt, das die rechte Konstituente bildet. Diese Organisationsform ist fuer den Parsing - Algorithmus optimal.

Die Bedeutung der weiteren Datenstrukturen, die mit der Grammatik zusammenhaengen, wie TYPPRODUKTION, TYPWORTKETTE, TYPSYNTAGMATISCH und TYPPARADIGMATISCH ergeben sich nach obiger Erlaeuterung wohl von selbst. Hingewiesen sei noch auf die beiden Strukturen TYPBASIS und TYPPHRASE : Sie stellen ein Aequivalent zum Parsing - ARRAY des Cooke - Algorithmus dar.

Wie man sieht, sind alle wesentlichen Daten dynamisch organisiert. Wir haben uns zu dieser Art der Datenverwaltung entschlossen, weil sie uns dem Problem angemessen erscheint und ausserdem bedeutend speicherplatzsparender ist als eine statische Loesung, wie sie z.B. im ALGOL - HASE vorliegt.

Um die Organisationsform der Grammatik - Daten noch einmal zu verdeutlichen, ist im Abschnitt 1 exemplarisch eine Grammatik in ihrer programminternen Realisierung mit allem "Drum und Dran" grafisch dargestellt.

1. Grafische Darstellung und Erlaeuterung
   Exemplarisch : ein Nonterminalbaum

Folgende einfache Grammatik wurde als Eingabe verwendet:

```
NONTERMINALS
S                0
NPN              2
DET              5
NN               4
EN               4
V                1
.
REGELN
NPN         <-- DET          + EN
NPN         <-- DET          + NN
S           <-- EN           + V
S           <-- NPN          + V
.
LEXIKON
V           <-- BELLT
V           <-- LAEUFT
DET         <-- DAS
DET         <-- DER
DET         <-- DIE
NN          <-- HUND
EN          <-- EUMEL
EN          <-- HANS
.
```

Die Zahlen in ">> <<" geben die symbolischen Adressen im dynamischen Datenbereich an. Die RECORDS wurden in der Reihenfolge ihrer Erstellung nummeriert. Das RECORD Nummer 1 wird zur Initialisierung des NONTERMINALZGR in der Prozedur INITIA erstellt. Nummer 2 bis 6 werden beim Einlesen der Nonterminale in der Prozedur LIESVOKABELNONTERMINAL erstellt und in den binaeren Baum eingehaengt. Die RECORDS 7 bis 14 repraesentieren die binaren Regeln, die erstens unter der rechten Konstituente [RECORDS vom Typ TYPREGEL] und zweitens unter ihrem Konstitut [RECORDS vom Typ TYPPRODUKTION] eingehaengt sind. Die RECORDS 15 bis 22 werden bei der Erstellung des Terminalbaumes an die syntaktischen Kategorien der Terminale gehaengt. Die Verweise der Art "<'HANS'>" sollen Zeiger auf das RECORD dieser Terminale im Terminalbaum darstellen. Auf die Darstellung des Terminalbaumes wurde verzichtet, weil es sich dabei um eine weniger komplexe Struktur handelt als bei dem Nonterminalbaum.

```
>>  1 << TYPNONTERMINAL
            NTNAME : 'S'
          NTSTUFE : 0
          NTREGEL : NIL
       NTPRODREGEL : < 12 >
    NTPRODTERMINALS : NIL
        NTRECHTER : < 6 >
         NTLINKER : < 2 >
   NTTRANSFORMATION : NIL
      NTTRANSLATOR : FALSE
```

```
>>  2 << TYPNONTERMINAL
            NTNAME : 'NPN'
          NTSTUFE : 2
          NTREGEL : NIL
       NTPRODREGEL : < 8 >
    NTPRODTERMINALS : NIL
        NTRECHTER : NIL
         NTLINKER : < 3 >
   NTTRANSFORMATION : NIL
      NTTRANSLATOR : FALSE
```

```
>>  6 << TYPNONTERMINAL
            NTNAME : 'V'
          NTSTUFE : 1
          NTREGEL : < 11 >
       NTPRODREGEL : NIL
    NTPRODTERMINALS : < 15 >
        NTRECHTER : NIL
         NTLINKER : NIL
   NTTRANSFORMATION : NIL
      NTTRANSLATOR : FALSE
```

```
>>  3 << TYPNONTERMINAL
            NTNAME : 'DET'
          NTSTUFE : 5
          NTREGEL : NIL
       NTPRODREGEL : NIL
    NTPRODTERMINALS : < 17 >
        NTRECHTER : < 4 >
         NTLINKER : NIL
   NTTRANSFORMATION : NIL
      NTTRANSLATOR : FALSE
```

```
>>  4 << TYPNONTERMINAL
            NTNAME : 'NN'
          NTSTUFE : 4
          NTREGEL : < 9 >
       NTPRODREGEL : NIL
    NTPRODTERMINALS : < 20 >
        NTRECHTER : NIL
         NTLINKER : < 5 >
   NTTRANSFORMATION : NIL
      NTTRANSLATOR : FALSE
```

```
>>  5 << TYPNONTERMINAL
            NTNAME : 'EN'
          NTSTUFE : 4
          NTREGEL : < 7 >
       NTPRODREGEL : NIL
    NTPRODTERMINALS : < 21 >
        NTRECHTER : NIL
         NTLINKER : NIL
   NTTRANSFORMATION : NIL
      NTTRANSLATOR : FALSE
```

```
>> 12 << TYPPRODUKTION
        PRREGEL : < 11 >
     PRNAECHSTER : < 14 >

>> 14 << TYPPRODUKTION
        PRREGEL : < 13 >
     PRNAECHSTER : NIL

>> 11 << TYPREGEL
        RGKONSTITUT : <  1 >
RGRECHTEKONSTITUENTE : <  6 >
 RGLINKEKONSTITUENTE : <  5 >
        RGNAECHSTER : < 13 >

>> 13 << TYPREGEL
        RGKONSTITUT : <  1 >
RGRECHTEKONSTITUENTE : <  6 >
 RGLINKEKONSTITUENTE : <  2 >
        RGNAECHSTER : NIL

>>  8 << TYPPRODUKTION
        PRREGEL : <  7 >
     PRNAECHSTER : < 10 >

>> 10 << TYPPRODUKTION
        PRREGEL : <  9 >
     PRNAECHSTER : NIL

>> 15 << TYPWORTKETTE
     WKTERMINAL : <'BELLT'>
    WKNAECHSTER : < 16 >

>> 16 << TYPWORTKETTE
     WKTERMINAL : <'LAEUFT'>
    WKNAECHSTER : NIL

>> 17 << TYPWORTKETTE
     WKTERMINAL : <'DAS'>
    WKNAECHSTER : < 18 >

>>  9 << TYPREGEL
        RGKONSTITUT : <  2 >
RGRECHTEKONSTITUENTE : <  4 >
 RGLINKEKONSTITUENTE : <  3 >
        RGNAECHSTER : NIL

>> 18 << TYPWORTKETTE
     WKTERMINAL : <'DIE'>
    WKNAECHSTER : < 19 >

>> 19 << TYPWORTKETTE
     WKTERMINAL : <'DER'>
    WKNAECHSTER : NIL

>> 20 << TYPWORTKETTE
     WKTERMINAL : <'HUND'>
    WKNAECHSTER : NIL

>> 21 << TYPWORTKETTE
     WKTERMINAL : <'EUMEL'>
    WKNAECHSTER : < 22 >

>>  7 << TYPREGEL
        RGKONSTITUT : <  2 >
RGRECHTEKONSTITUENTE : <  5 >
 RGLINKEKONSTITUENTE : <  3 >
        RGNAECHSTER : NIL

>> 22 << TYPWORTKETTE
     WKTERMINAL : <'HANS'>
    WKNAECHSTER : NIL
```

## 2. Einlesen der Syntax - Datei

Beim Einlesen der Syntax-Datei, deren Format oben beschrieben wurde, sind die Prozeduren LIESNONTERMINAL, LIESTERMINAL, LIESSCHLUESSELWORT, SETZNONTERMINAL, LIESVOKABELNONTERMINAL, LIESTRANSLATION, LIESREGEL, ERSTELLEREGEL, LIESLEXIKON, ERSTELLELEXIKON, LIESFRAGEWORT, LIESTRANSFORMATION und SYNTAXLESEN beteiligt. Dieser Teil des Programmes, ist im Gegensatz zum urspruenglichen ALGOL-Programm, ziemlich lang geworden. Das liegt daran, dass die Datenstrukturen des jetzigen Programmes komplexer, und damit auch komplizierter zu handhaben sind, und ausserdem die Programmiersprache PASCAL in Bezug auf STRING-Verarbeitung nicht die Maechtigkeit des hiesigen ALGOL-Dialekts besitzt. Daraus ergab sich auch das Problem: wie sollen Strings abgespeichert werden? Es gibt im Standard-PASCAL im wesentlichen zwei Methoden zur Loesung dieses Problems:

a) Man speichert eine Zeichenkette in ein ARRAY fester Laenge. Das fuehrt allerdings dazu, dass der Speicherplatz kaum ausgenutzt wird.

b) Man speichert eine Zeichenkette in einer linearen Liste, z.B.

```
TYPSTRING = RECORD
                INFORMATION : ALFA;
                NAECHSTER : ^TYPSTRING
            END;
```

Bei dieser Loesung wird allerdings viel Platz fuer die Verweise benoetigt, und es ergibt sich die Notwendigkeit, Bearbeitungsprozeduren zu schreiben (fuer Aufbau, Vergleiche und Muellverwaltung der Zeichenketten). Aus Gruenden der Effektivitaet (Laufzeit, Speicherplatz und Programmieraufwand) haben wir uns fuer Loesung a) entschieden.

### 2.1. Initialisierung

In der Prozedur INITIA werden die Variablen initialisiert, die in einer INITPROCEDURE nicht besetzt werden koennen (dynamische und gepackte Strukturen). Zusaetzlich wird die Kopfzeile des Programmes ausgeschrieben und die Prozedur SYNTAXLESEN aufgerufen, womit die Anfangsphase des Programmes beendet ist.

Es werden folgende Variablen in INITIA initialisiert:

a) Das LEERPARADIG-ARRAY, das in der Prozedur ERSTELLELEXIKON benoetigt wird.

b) Die beiden Zeiger STERNCHENTERMINAL und STERNCHENNONTERMINAL, die stellvertretend fuer alle Zeiger auf Terminale und Nonterminale stehen (siehe auch 6.).

c) Das Startsymbol "S" wird als erstes Nonterminal in den Nonterminalbaum gehaengt, d.h. der NONTERMINALZGR zeigt direkt auf "S".

### 2.2. Einlesen des nonterminalen Vokabulars

Je eine Zeile der Sektion "Nonterminals" der Syntax-Datei wird mit Hilfe der Prozedur LIESVOKABULARNONTERMINAL verarbeitet. Diese Prozedur liest ein Nonterminal und die zugehoerige Dependenzstufe, erzeugt dann mit Hilfe von SETZNONTERMINAL eine neue Variable des Typs TYPNONTERMINAL , besetzt sie entsprechend und plaziert sie im Nonterminalbaum.

## 2.3. Einlesen der Translatoren

Die Prozedur LIESTRANSLATION liest ein Nonterminal und setzt das Feld NTTRANSLATOR des entsprechenden RECORDs auf TRUE.

## 2.4. Einlesen der Transformationsregeln

Als Datenstruktur sind die Transformationsregeln (die lediglich bei einem Programm mit Semantikteil sinnvoll sind) folgendermassen verwirklicht: Wenn ein Nonterminal Transformationsausloeser ist, so zeigt der Zeiger NTTRANSFORMATION dieses RECORDs auf das erste Glied einer Kette, die aus RECORDs des Typs TYPTRANSFORMATION bestehen. Diese enthalten einen Verweis auf das Substitut und den Substituenden. Das Einlesen und Einsetzen je einer Transformationsregel wird von der Prozedur LIESTRANSFORMATION erledigt.

## 2.5. Einlesen der binaeren Regeln

Die binaeren Regeln der Syntax werden wie folgt auf die Datenstruktur des Programmes abgebildet: Ist ein Nonterminal rechte Konstituente einer Regel, so zeigt NTREGEL auf eine Kette von RECORDs vom Typ TYPREGEL, in denen ein Verweis auf die linke Konstituente, einer auf das Konstitut und ein Rueckverweis auf die rechte Konstituente eingetragen ist. Diese Organisationsform, das Ordnen der Regeln nach der rechten Konstituente, ist fuer den Parsing-Algorithmus optimal. Da innerhalb des Programmes (in dieser Version) auch Saetze generiert werden sollen, existiert zusaetzlich bei jedem Nonterminal, das Konstitut einer Regel ist, ein Verweis NTPRODREGEL auf eine Kette von RECORDs vom Typ TYPPRODUKTION. Diese tragen Verweise auf die Regeln. Dadurch kann auch vom Konstitut ausgehend eine Regel effizient aufgesucht werden.

Eingelesen wird je eine Regel von der Prozedur LIESREGEL, die den logischen Wert TRUE liefert, falls alle bei der Regel angegebenen Nonterminale bekannt sind. Die drei Rueckgabeparameter koennen dann an die Prozedur ERSTELLEREGEL uebergeben werden, die die Regel dann in der oben beschriebenen Art einsetzt.

## 2.6. Einlesen des terminalen Vokabulars und der terminierenden Regeln

Das terminale Vokabular wird aehnlich wie das nonterminale in einem binaeren Baum gespeichert. Da jedem Terminal verschiedene lexikalische Regeln und semantische Beziehungen zugeordnet sein koennen (man denke nur an das Wort "eben") enthaelt jedes Terminal-RECORD einen Verweis TLSEMANTIK auf eine Kette von RECORDs des Typs TYPSEMANTIK. In diesen sind die syntaktischen und semantischen Beschreibungen gespeichert.

In der Sektion "Lexikon" der Syntax-Datei wird das terminale Vokabular eingefuehrt, indem die terminierenden Regeln angegeben werden. Gelesen wird je eine Regel von der Funktion LIESLEXIKON, die auch prueft, ob das angegebene Nonterminal vorhanden ist. Eingetragen wird das Terminal und die Regel von der Prozedur ERSTELLELEXIKON. Hier wird wieder, aehnlich wie beim Eintrag der binaeren Regeln, zusaetzlich zum Verweis in die

reduzierende Richtung [SMKATEGORIE], also vom Terminal zum Nonterminals, ein Verweis in die expandierende Richtung eingetragen, um das Generieren von Saetzen zu unterstuetzen.

2.7. Einlesen der Fragewoerter

Die Fragewoerter (deren Angabe nur bei einem Programm mit Semantikteil sinnvoll ist) werden von der Prozedur LIESFRAGEWORT eingelesen, die dann das boolsche Feld SMFRAGEWORT in den entsprechenden Semantik-RECORDs auf TRUE setzt.

3. Prozedur KOMMANDO und Grammatikausgabe

Die Prozedur KOMMANDO ist die Schaltstelle zwischen den verschiedenen Moduln des Programms. Nach dem Starten des Programms und dem Spezifizieren der Ein/Ausgabe-Dateien kommt der Benutzer automatisch in den KOMMANDO-Modus. Aus dem KOMMANDO-Modus ist der Aufruf der Prozeduren ANALYSE (siehe 4.), GENERIERE (siehe 5.) und ERWEITERE (siehe 6.) und der damit verbundene Uebergang in die entsprechenden Modi moeglich. Ausserdem kann die Ausgabe des gegenwaertigen Standes der Grammatik auf eine Datei erreicht werden (siehe 3.1.). Die Beendigung des Programms erfolgt nach dem Eingeben eines Punktes.

3.1. Ausgabe der Grammatik

Die Ausgabe der Grammatik erfolgt auf die zu Programmstart spezifizierte Datei des Dateibezeichners SYNTAXAUS. (Es kann z.Z. nur eine Grammatik pro Programmlauf ausgegeben werden.) Die Reihenfolge der Ausgabe ist gleich der der Eingabe einer Grammatik, sodass es dem Benutzer moeglich ist, Dateien eines frueheren Programmlaufs wiedereinzulesen.

Die Prozedur AUSGABESYNTAX steuert die Ausgabe und ruft die entsprechenden Prozeduren in der Reihenfolge der Bezeichner des Typs TYPSEKTIONEN auf.

Wir haben bei einigen Prozeduren darauf verzichtet, die Nonterminale bzw. Terminale in alphabetischer Reihenfolge auszugeben, weil dieses beim Wiedereinlesen der Datei zu einem extrem unausgewogenen binaeren Baum (Kette!) fuehren und sich besonders bei groesseren Grammatiken sehr unvorteilhaft auf die Suchzeit auswirken wuerde.

Das Ausschreiben der Nonterminale, der Translatoren unter den Nonterminalen, der Transformationen und der Fragewoerter unter den Terminalen erfolgt jeweils durch einfache rekursive Prozeduren.

Bei der Ausgabe der Regeln wird die alphabetische Reihenfolge der Konstitute eingehalten. (Beim Wiedereinlesen ist die Reihenfolge der Regeln fuer die Anordnung des binaeren Baumes belanglos.) Da die Regeln jeweils unter ihrer rechten Konstituenten eingeordnet sind, bedarf es eines mehrfachen Durchsuchen des Baumes (jeweils einmal pro Nonterminal). Die Prozedur SCHREIBREGELN steigt rekursiv in den Baum der Nonterminale und ruft fuer jedes Nonterminal die Prozedur SCHREIBFUERKONSTITUT auf, welche alle Regeln durchsucht, ob dieses Nonterminal als Konstitut auftritt und es dann ausschreibt.

# 4. ANALYSE

## 4.1. Interaktives Einlesen, Aufbau der Basiskette

Um einen Satz zu analysieren, muss dieser zunaechst geeignet vom Programm aufgenommen werden.Dieses geschieht interaktiv, zeilenweise ueber das TTY [ SATZEINLESEN ]. Jedes Wort wird einzeln in ein ARRAY OF CHAR eingelesen und nur der Verweis auf den RECORD dieses Bezeichners wird ueberseben [ LIESTERMINAL , SUCHTERMINAL ]. Falls der Bezeicher nicht bekannt ist wird mit einer Fehlermeldung abgebrochen [ lokale BOOLEANvariable FEHLER , FEHLERTERMINAL , DISPOSE ]. Um die Struktur der Satzes zu erhalten, wird eine Kette von BASISRECORDs erstellt, welche den oben genannten Verweis tragen und einen weiteren, welcher Grundlage zur Verarbeitung ist. Dieser zweite Verweis zeigt auf PhrasenRECORDs. Sie tragen Information, welche sich aus dem Lexikoneintrag des Terminals ergeben [ EINBAU , ERSTELLEPHRASEN ]. Speziell bei mehrdeutigen Terminals werden entsprechend viele RECORDs als Kette angehaengt; auf die PhrasenRECORDs wird in 4.2 weiter eingegangen. Nach dem Einlesen ist somit alle notwendige Information in einer geeigneten (siehe 4.2) Struktur zusammengetragen. Diese vollstaendige Abstraktion erlaubt auf Buchstaben und Worte nicht weiter einzugehen.

## 4.2. Abarbeiten und Parsen

Analysiert wird nach einem Parsing-Algorithmus (siehe Teil I) [PARSE]. Dieser Algorithmus arbeitet ausschliesslich auf PhrasenRECORDs. Er geht von vorhandenen RECORDs aus und erstellt nach Informationen aus den binaeren Regeln der Grammatik neue RECORDs, die er weiterverarbeitet. Der Algorithmus hinterlaesst nur vielfaeltig verknuepfte RECORDs, die anschliessend ausgewertet werden koennen. Beim Suchen passender Regeln wird vom rechten Konstituenten ausgegangen (siehe auch 2.1). Durch den Vorverweis [ PHVORGAENGER ] wird die Information, wie weit das Konstitut den Satz ueberdeckt, ueberseben. Dieser Verweis zeigt auf den BasisRECORD dessen Phrasen linke Konstituenten sein koennen. Wird der Satz durch ein Konstitut nach links voellig ueberdeckt, so ist der Verweis NIL.

Wie in 4.1 erwaehnt, werden den BasisRECORDs PhrasenRECORDs zugeordnet. Die Information fuer Semantik, Kategorie und Prioritaet werden aus den SemantikRECORDs der Terminale entnommen. Kategorie und Priorietaet sind somit direkt und mittels der SemantikRECORDs indirekt spezifiziert. Die Konstituentenverweise sind NIL, weil keine Regel angewendet wurde, der Naechster- und vor allem der Vorgaengerverweis sind richtig besetzt.

Die Abarbeitung erfolgt Wort fuer Wort von links nach rechts. Bei jedem Wort werden a l l e moeglichen Konstitute gebildet, angefangen bei denen, die nur das Wort selbst erfassen, bis hin zu denen -falls existent- die den Satz von vorne bis einschliesslich des Wortes ueberdecken. Dabei werden alle neu gebildeten Phrasen als moegliche Konstituenten getestet. Durch dieses Vorgehen wird die gesammte Analyse in einem Druchgang ausgefuehrt, ueberfluessiges doppeltes Pruefen einer Moeglichkeit ist ausgeschlossen. Vorgesehen, aber noch nicht ausgenutzt, ist ueber die Syntaxpruefung hinaus eine Semantikpruefung (siehe auch 7.).

### 4.3. Aussabe

Als Aussabe folst die Ansabe der Vieldeutiskeit des Satzes. Dazu sucht ERGEBNIS alle Phrasen des letzten BasisRECORD ab, ob sie Ueberdeckunsen des sanzen Satzes darstellen und ob sie von der Kategorie her ein Satz sind. Zusetzlich koennen eine Wesetabelle, sowie die Dependenz- und/oder Konstituentenstucktur ausseseben werden ( siehe II ). WEGETABELLE arbeitet in zwei REPEAT-Schleifen alle PhrasenRECORDs ab und schreibt die sesamte Information formatiert aus. Dabei werden die einzelnen Phrasen durchnummeriert,so dass die Bildung von ersichtlich wird. Die Prozeduren DEPENDENSSTRUKTUREN und KONSTITUENTENSTRUKTUREN sind links- und rechtsrekursiv.

## 5. Prozedur GENERIERE

Das Format der Eingabe und die Bedeutung der Schalter ist aus der Benutzeranleitung zu ersehen und wird als bekannt vorausgesetzt. Die Schalter werden programmintern in dem SET GENSCHALTER, sowie in den INTEGER Variablen GENREKURSIONSGRAD und GENTERMINATIONSGRAD repraesentiert.

Der Kern der Generierung ist die rekursive Prozedur SATZGENERIERUNG. Sie wird zu Beginn jeder Generierung mit dem Startsymbol (NONTERMINALZGR <=> "S") aufgerufen und ruft sich, wenn eine nicht terminierende Regel gewaehlt worden ist, zuerst fuer die linke Konstituente und dann fuer die rechte Konstitutente selber auf. Bei terminierenden Regeln traegt sie das gefundene Wort in eine Satzkette ein, auf deren Anfang die Variable GENSATZZGR zeigt.

Fuer die Satzgenerierung werden bei jedem Nonterminal besondere Informationen gespeichert. So haengt an jedem NONTERMINALRECORD, falls in der Grammatik vorhanden, erstens eine Kette von Verweisen auf Regeln [NTPRODREGELN], deren Konstitut dieses Nonterminal ist, und zweitens eine Kette von Verweisen auf Terminale [NTPRODTERMINAL], deren syntaktische Kategorie dieses Nonterminal ist. Diese Ketten werden waehrend des Einlesens der Grammatik aufgebaut und dienen ausschliesslich zur optimalen Durchfuehrung (in Bezug auf Laufzeit) des Generierens (siehe auch 2.5).

Die Prozedur SATZGENERIERUNG baut sich im einzelnen wie folgt auf: Nachdem festgestellt worden ist, ob dieses Nonterminal ueberhaupt weiter expandiert werden kann, wird bei zufaelliger Generierung bestimmt, ob eine nicht terminierende Regel verwendet werden soll. Das Setzen der boolschen Variablen LREGELNEHMEN wird entschieden in Abhaengigkeit einer Zufallszahl und dem Wert der Variablen GENTERMINATIONSGRAD. Im folgenden werden bei zufaelliger Generierung nur die Anzahl [LANZAHL] der vorhandenen Moeglichkeiten gezaehlt, waehrend sonst alle moeglichen Regeln auf der Benutzerstation ausgegeben werden. Die Wahl der Regel erfolgt entweder durch eine Eingabe des Benutzers oder durch Zufall. Bei zufaelliger Generierung und der Wahl einer rekursiven Regel wird per Zufall und in Abhaengigkeit der Variablen GENREKURSIONSGRAD entschieden, ob diese Regel verwendet wird. Wenn nicht, erfolgt die Auswahl einer neuen Regel. Da die Variable GENREKURSIONSGRAD nur Werte zwischen 1 und 9 annehmen darf, die Zufallszahl aber zwischen 1 und 10, terminiert diese Schleife.

## 6. Prozedur ERWEITERN

Die Form und die Bedeutung der einzelnen Eingaben werden im folgenden als bekannt vorausgesetzt (siehe II). Um es dem Benutzer zu gestatten, verschiedene Regeln oder Lexikoneintraege zu loeschen oder zu suchen (<=> auf der Benutzerstation auszugeben, falls vorhanden), haben wir zwei besondere Variablen [STERNCHENTERMINAL, STERNCHENNONTERMINAL] eingefuehrt. Nur bei neuen Eintraegen in das Lexikon oder Hinzufuegen von Regeln muessen die Eingaben genau spezifiziert werden. Jeweils vor dem Setzen eines neuen Eintrags wird geprueft, ob es sich um das STERNCHENTERMINAL bzw. STERNCHENNONTERMINAL handelt. Wenn Ja, wird eine Fehlermeldung ausgegeben und das Einsetzen abgebrochen.

Beim Loeschen von Eintraegen werden die entsprechenden RECORDS nur aus den Verweisketten ausgehaengt. Sie verbleiben waehrend des restlichen Programmlaufes als "Leichen" im Hauptspeicher, weil es bei unserer hiesigen PASCAL-Implementation nicht moeglich ist, spezielle Teile des dynamischen Datenbereichs wieder freizugeben. Ausserdem muessen beim Loeschen auch alle Verweise auf diese Eintraege beseitigt werden [LOESCHEREGEL, LOESCHEIMLEXIKON]. Dieses erweist sich besonders bei den fuer die Generierung angelegten Verweisen als sehr aufwendig.

Das Loeschen von Nonterminalen ist nicht gestattet, weil uns der dafuer benoetigte Aufwand (Umbau des binaeren Baumes der Nonterminalen, Loeschen aller Verweise auf dieses Nonterminal und aller Regeln, in denen es vorkommt) als nicht gerechtfertigt erschien.

Ebenso ist das Loeschen von Terminalen verboten. Allerdings koennen alle mit einem Terminal assoziierten terminierenden Regeln geloescht werden, so dass das Terminal nicht mehr benutzt werden kann.

## 7. Erweiterungsmoeglichkeiten

Durch konsequente modulare Programmierung sind Erweiterungen verhaeltnismaessig einfach einzusetzen. In vielen Faellen duerfte dieses besonders schnell moeglich sein, weil sowohl die Datenstrukturen als auch die Prozeduren weitgehend vorhanden sind. Speziell die Einlese und Ausgabeprozeduren sind vielfaeltig verwendbar gestaltet. Die Laenge der Terminale und Nonterminale sind Konstanten und muessen im Bedarfsfall nur an einer Stelle im Programm geaendert werden. Dabei bleibt die geeignete Formatierung der Ausgabe erhalten.

Wie bereits in 4.2 erwaehnt ist eine Erweiterung um eine semantische Analyse vorbereitet. Diese Vorbereitung umfasst nicht nur den Prozeduraufruf sondern im besonderen die Datenstrukturen. Es wurde davon ausgegangen, dass das Programm in einer Version als Semantik - Parser ausgebaut wird. An Jeden Semantik - RECORD der Terminal - RECORDs sind Verweise auf Wortketten von Verweisen auf Terminale vorhanden. Hier koennen alle synonymen, hyponymen, komplenymen und antonymen Woerter der Lexikons eingetragen werden. Bei Vieldeutigkeit des Lexikoneintrages gilt dieses fuer jede Deutigkeit. Entsprechendes gilt fuer die Bedeutung inhaerenter Eigenschaften.

Neben dieser festen Struktur ist eine dynamische Struktur fuer syntasmatische Eisenschaften vorsesehen. Wenn diese Strukturen nicht ausreichen, sibt es die Moeslichkeit, sowohl die Mense von paradismatischen Eisenschaften zu erweitern, als auch durch weitere Felder im Semantik - RECORD - sesebenfalls mit entsprechenden Ketten - zusaetzliche, anders seartete Informationen hinzuzufuesen.

Die interaktiven Faehiskeiten des Prosrammes koennten noch erweitert werden, so dass es mehr Moeslichkeiten der Beeinflussuns des Prosrammablaufes seben wuerde. Besonders fuer spezielle Verwendunsszwecke waere eine noch detalliertere Aussabe sowohl bezueslich der Analyse als auch der Synthese von Saetzen moeslich, zum Beispiel um einem Anfaenser den Zusans zur Syntax und Semantik einer besrenzten Sprache zu erleichtern. Interessant waere auch die Moeslichkeit, direkt durch die Einsabe von Saetzen die Grammatik zu erweitern (allerdinss ist dieses schon von der Theorie her problematisch !).

Bisher erschienene Institutts-Mitteilungen:

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

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

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

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

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

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

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

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

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

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

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

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

13 MEYER, A.
EINFUEHRUNG IN DAS RETRIEVALSYSTEM DER INFORMATIKBIBLIOTHEK HAMBURG
II/75, 17 P

14 WENDT, S.
FUNKTIONSBESCHREIBUNG DES INTEGRIERTEN PROZESSORBAUSTEINS INTEL 800
III/75, 20 P

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

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

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

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

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

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

21 NAGEL, H.-H.
PASCAL FOR THE DECSYSTEM-10, EXPERIENCE AND FURTHER PLANS.
XI/75, 7 P

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

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

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

25 BORN, H. U., KUEHL, G., STEIMER, D.
EVALUATION NETWORKS ALS BESCHREIBUNGSMITTEL FUER RECHENSYSTEME
I/76, 209 P

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

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

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

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

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

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

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

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

34 NEBEL, B., PRETSCHMER, B.
ERWEITERUNG DES DECSYSTEM-10 PASCAL-COMPILERS UM EINE MOEGLICHKEIT ZUR ERZEUGUNG EINES POST-MORTEM-DUMP
VI/76, 27 P

35 HEIBEY, H.-W., LUTTERBECK, B., ROHLFS, S., TOEPEL, M.
INFORMATIONSVERAENDERUNGEN BEIM EINSATZ DER DATENVERARBEITUNG
IX/76, 201 P