Automatisches Zeichnen von Diagrammen

Eidgenössische
Technische
Hochschule
Zürich

*Berichte
des
Instituts für
Informatik*

Karl Bucher

Automatisches Zeichnen von Diagrammen

Juli 1977 *23*

Eidgenössische
Technische
Hochschule
Zürich

*Berichte
des
Instituts für
Informatik*

Karl Bucher

Automatisches Zeichnen von Diagrammen

Abstract

Several techniques for automatic drawing of a wide class of diagrams are presented and evaluated. The line drawings that can be processed are recursively structured: subdiagrams consist of several vertically connected parallel sections. Each section consists of subdiagrams and atoms that are horizontally connected. Connections need not be visible in the drawing. Atoms are arbitrary figures and may contain text. Syntax diagrams are an example of the above class.
The plotter programs must satisfy the following demands:
- Comfortable input and modification: short, readable, easy and safe commands expressed in terms of high-level pictorial properties, rather than in terms of coordinates.
- Flexible output: aesthetic, useful diagrams.

A graph rewriting system controlled by a sequential notation generates the diagram graphs. The two-dimensional coordinate assignment problem is separated into two different one-dimensional problems. Algorithms for solving these problems and a language for describing the input and output of each specific plotter program are presented. A meta program can then automatically transform the above description into the plotter program.

Zusammenfassung

Techniken fuer das automatische Zeichnen von Diagrammen wurden hier entwickelt. Die behandelbaren Strichzeichnungen sind rekursiv strukturiert: Unterdiagramme bestehen aus vertikal verbundenen parallelen Abschnitten. Jeder Abschnitt besteht aus Unterdiagrammen und Atomen, welche untereinander horizontal verbunden sind. Dabei brauchen Verbindungen keine sichtbaren Linien der Zeichnung zu sein. Atome sind beliebige Figuren, die Text enthalten duerfen. (Beispiel: Syntaxdiagramme).
Die erwuenschten Zeichnungsprogramme sollen folgende Anforderungen erfuellen:
- Bequeme Eingabe- und Modifikationsbefehle: kurze, lesbare, einfache und sichere Befehle in einer hoeheren bildorientierten Sprache. Keine Koordinatenmanipulationen.
- Flexible Ausgabe: aesthetische und praktische Diagramme.

Ein Graphenersetzungssystem, das durch eine sequentielle Notation gesteuert wird, erzeugt die Diagrammgraphen. Das zweidimensionale Koordinatenzuweisungsproblem wird in zwei eindimensionale getrennt. Fuer die Loesung dieser Probleme werden Algorithmen angegeben. Eine Sprache wird vorgestellt, in der Ein- und Ausgabe fuer jedes spezifische Zeichnungsprogramm P beschrieben werden koennen. Ein Metaprogramm ist dann in der Lage, diese Beschreibung automatisch in das Programm P umzusetzen.

Adresse des Verfassers:

Institut fuer Informatik
ETH-Zentrum
CH-8Ø92 Zuerich

Inhaltsverzeichnis
==========

Ø. Einfuehrung

Ø.1 Vorwort

Dieser Bericht ist ein Auszug aus der Dissertation [2] ueber das automatische Zeichnungen von Diagrammen, welche ich unter der Leitung von Prof. Dr. P. Laeuchli ausgefuehrt habe.
Aus jener Arbeit wurde hier insbesondere weggelassen:
- Die Besprechung der Probleme, welche sich aus der Verwendung der gewoehnlichen Backus-Naur-Form fuer die Eingabe ergeben.
- Die Einbettungsmoeglichkeiten der beschriebenen Programme in eine interaktive Umgebung.
- Der Vergleich mit andern Arbeiten ueber automatisches Erzeugen von Syntaxdiagrammen.
- Die Beschreibung von Hilfsprogrammen fuer die Ausgabe von Bildern.
- Eine Uebersicht ueber Beziehungen zu aehnlichen Problemstellungen und Arbeiten samt dem zugehoerigen Literaturverzeichnis.

Ø.2 Einleitung

Diese Arbeit handelt vom automatischen Zeichnen einer Klasse von Diagrammen. Die betrachteten Diagramme sind mit Text und Symbolfiguren versehene Netzwerke, deren Zweck es ist, ´´gelesen´´ zu werden. D.h. solche Diagrammzeichnungen dienen z.B. der Darstellung von Folgen von Informationspartikeln. Wegen der Lesbarkeit, die von vielen dieser Diagramme verlangt wird, werden hier diejenigen Diagramzeichnungen besonders hervorgehoben, deren Verbindungslinien horizontal und vertikal verlaufen. Eine weitere wichtige Eigenschaft von solchen Zeichnungen ist die Planaritaet. Diese beiden Eigenschaften haben die hier entwikkelten Methoden beeinflusst, sind aber nicht Bedingung fuer ihre Anwendbarkeit. Diagramme, welche mit diesen Methoden dargestellt werden koennen, erfuellen folgendes Aufbauprinzip:
- Die Netzwerkkanten koennen von ´´Merkmalen´´ unterbrochen sein. Merkmale bestehen fuer jede Diagrammklasse aus einer Menge von fest vorgegebenen Figuren (Atome), welche Text enthalten duerfen.
- Die Diagramme sind rekursiv durch miteinander verbundene Unterdiagramme gegliedert:
  A) Ein Atom ist ein Unterdiagramm.
  B) Ein Sektor ist eine serielle Anordnung von Unterdiagrammen. Die Anordnung ist durch Verknuepfungslinien definiert, welche in der Zeichnung aber nicht unbedingt sichtbar sein muessen. Dagegen koennen die Unterdiagramme durch beliebige Anschlussstuecke verbunden sein. Zum Sektor gehoert der ganze Raum, der innerhalb des kleinsten Rechtecks mit vertikalen und horizontalen Kanten liegt, welches alle Unterdiagramme der Serie umschreibt.
  C) Ein (Unter-)Diagramm besteht aus einer Folge paralleler Sektoren, die sich vernuenftigerweise nicht ueberschneiden sollten. Punkte aus verschiedenen (benachbarten) Sektoren duerfen vertikal verknuepft sein. Eine Verknuepfung ist eine sicht- oder unsichtbare Verbindungslinie.

Die Motivation fuer diese Arbeit war der ausgewiesene Bedarf an Computerprogrammen fuer das automatische Zeichnen von speziellen Klassen von Diagrammen aus obigem Rahmen. Besonders erwaehnt sei die Klasse der Syntaxdiagramme, mit welchen die Syntax einer formalen Sprache durch Bilder definiert wird.

Die Ziele dieser Arbeit sind die folgenden:

1) Bestimmen einer Eingabeform: Die zu zeichnenden Figuren muessen mit moeglichst kleinem Aufwand beschrieben und veraendert werden koennen. Geometrische Operationen wie Einfuegen oder Verschieben von Punkten und Linien sind somit ausgeschlossen. Es sollen also Eingabe- und Modifikationsoperationen von hoechstmoeglicher Stufe verfuegbar gemacht werden.
2) Bestimmen einer Ausgabeform: Trotz der einfachen Eingabe soll der Anwender diejenigen Netzformen ausdruecken koennen, die seine Diagrammklasse vorwiegend charakterisieren.
3) Bestimmen von Einbettungsalgorithmen: Der Layout mit allen Koordinatenberechnungen soll ausschliesslich vom Darstellungsprogramm geleistet werden. Das Resultat dieser Berechnungen muss publikationswuerdig und -faehig sein. Dies bedeutet, dass einige aesthetische und praktische Forderungen an die Zeichnungen erfuellt sein muessen, wie etwa gleichmaessige Platzausnuetzung.
4) Bestimmen des erforderlichen Programmaufbaus: Der Aufbau von Zeichnungsprogrammen, welche die vorigen drei Punkte beruecksichtigen, ist klar zu artikulieren. Die gewonnene Einsicht in den Aufbau solcher Programme soll zur systematischen Konstruktion von Darstellungsprogrammen fuer verschiedene Diagrammklassen verwendet werden.

Die aufgezaehlten vier Aufgaben werden unter der Voraussetzung angegangen, dass jeder Anwender sich fuer eine unveraenderliche Diagrammklasse interessiert, aus dieser jedoch ueber laengere Zeit eine grosse Anzahl von Zeichnungen erzeugen lassen will.

Das verfolgte Ziel wird im Laufe dieser Arbeit ueber mehrere Etappen erreicht. Die auftretenden Probleme bei den Punkten 1), 2) und 3) werden anhand des Beispiels der Syntaxdiagramme erlaeutert. Implementationen von Darstellungsprogrammen fuer derartige Diagramme werden erklaert. Insbesondere wird die Approximation der Diagramme durch Serie-Parallel-Graphen hervorgehoben. Dieses Graphenschema bestimmt durch die Definitionen A), B) und C) in versteckter Form alle hier betrachteten Graphiken.

In einem weiteren Schritt wird eine Sprache entworfen, mit der Ein- und Ausgabe jedes spezifischen Zeichenprogramms P beschrieben werden koennen. Diese Beschreibung kann mechanisch in ein solches Programm P umgesetzt werden. Ein Programm, welches diese Umsetzung automatisch ausfuehrt, wurde fuer horizontal-vertikal-Graphen und einen festen Satz von atomischen Figuren implementiert und erprobt.

Den theoretischen Hintergrund fuer das beschriebene Vorgehen bilden Modellansaetze der linguistischen Bildanalyse und -Synthese. Von diesen sind hervorzuheben:

- Das Modell von A.C. Shaw (s.[3]) mit seiner hierarchischen Gliederung der Bilder.
- Graphen- und Gewebegrammatiken, welche Graphen- bzw. Gewebemengen durch Graphenersetzungssysteme definieren.

In einem Zusammenhang steht diese Arbeit auch mit den diversen Anstrengungen zum automatischen Zeichnen von Flussdiagrammen, welche seit Beginn der 60-er Jahre unternommen wurden. Und schliesslich laesst sie sich auch mit Arbeiten aus dem Bereich des Computer Assisted Design vergleichen. Die entsprechenden Einordnungen sind ebenfalls in [3] zu finden.

1. Forderungen an die Zeichenprogramme

1.1 Anforderungen an die Ausgabe von Syntaxdiagrammen

An dieser Stelle sollen die Forderungen aufgezaehlt werden, welche man an die Ausgabe von Syntaxdiagrammen und deren Modifizierbarkeit stellen will. Dazu werden vorerst die Syntaxdiagramme definiert.

Ein Syntaxdiagramm ist ein planer zusammenhaengender Graph, dessen Kanten vertikal und horizontal verlaufen. Kanten koennen durch Kaestchen mit Textinhalt unterbrochen sein. Es sind genau zwei Punkte vom Grad eins vorhanden, naemlich Eingangs- und Ausgangspunkt des Diagramms.

Von einem abstrakten Diagramm wollen wir sprechen, wenn von einer Zeichnung der obigen Art nur kombinatorische, aber keine geometrische Information gegeben ist. D.h. von jedem Eckpunkt des Diagramms weiss man, in welcher Richtung er mit welchem Punkt verbunden ist. Ueber die Laenge der Kanten ist aber nichts bekannt.

Ein nivelliertes Diagramm soll ein abstraktes Diagramm sein, dessen Eckpunkte vertraegliche Hoehenkoordinaten besitzen. Vertraeglich heissen die Hoehenkoordinaten, wenn fuer beliebige Ausmasse der Atome noch Breitenkoordinaten angegeben werden koennen, sodass die resultierende Zeichnung kreuzungsfrei in der Ebene eingebettet ist.

Zwei Diagrammzeichnungen sollen zueinander aequivalent heissen, wenn sie dieselbe Teilsprache definieren.

Spezielle Eigenschaften von Syntaxdiagrammen

Es gibt offenbar zu jeder Diagrammzeichnung D beliebig viele von D verschiedene Diagramme, welche mit D aequivalent sind. Darunter befinden sich alle jene Diagramme, welche aus D mittels Ersetzen von Leerwegen durch kompliziertere Leerwege entstehen. Mit einem Programm, welches nicht die Koordinaten aller Diagrammeckpunkte als Eingabe verlangt, werden nicht alle denkbaren Diagrammzeichnungen, sondern fuer jede Aequivalenzklasse nur eine Menge von Repraesentanten, darstellbar sein. Insbesondere wird man Leerwege nicht als beliebige Umwege formen koennen.
Diese Repraesentantenmenge sollte in der Menge der aequivalenten Diagramme so dicht sein, dass jedes vernuenftige Diagramm E ein einigermassen adaequates Gegenstueck in der Repraesentantenmenge findet, welches zu E aequivalent ist.
Damit haben wir als erste Forderung an den Output:

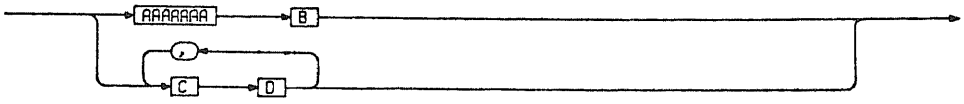
> Das vorgegebene (z.B. handgezeichnete) Bild eines Syntaxdiagramms soll moeglichst adaequat dargestellt werden koennen.

Ist diese Forderung mit einer speziellen Eingabe nicht fuer jedes Diagramm zu erfuellen, so muss das angegebene Diagrammbild aequivalent umschrieben werden.
An diese Umschreibung darf man die Forderung stellen, dass sie nicht wesentlich komplizierter sei als das vorgegebene Bild. D.h. fuer einfache Diagrammdefinitionen muessen auch einfache Diagrammzeichnungen

automatisch erzeugt werden koennen. Die Diagramme sollen also moeglichst einfach sein und moeglichst wenige Linien aufweisen.

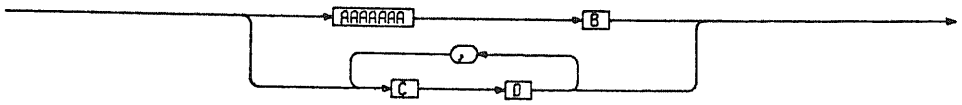
Ist ein nivelliertes Diagramm gegeben, so muss noch eine Breiteneinteilung fuer das Diagramm bestimmt werden. Dabei ist es z.B. leicht, eine ''linksbuendige'' Einbettung zu finden.

Ein Beispiel fuer eine linksbuendige Einbettung ist:



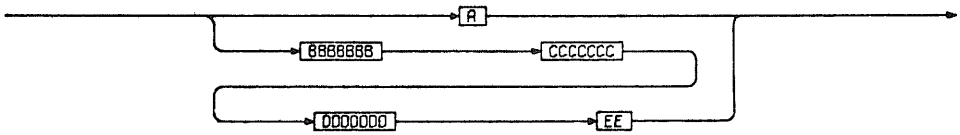
Da solche linksbuendige Diagramme u.U. nicht besonders gut lesbar sind, fordern wir vom Output eine ''ausgeglichene'' Breiteneinteilung.

Fuer obiges Beispiel waere das:



Die Breite einer zu publizierenden Diagrammzeichnung ist in der Regel vorgegeben durch die Breite des verfuegbaren Platzes auf der Buchseite. Dadurch ist die Summe aller Kaestchenlaengen auf jedem Niveau begrenzt.
Sehr lange Serien von Kaestchen oder anderen Diagrammelementen muessen also ''gebrochen'' werden koennen, damit das Diagramm auf der gegebenen Seite Platz findet.

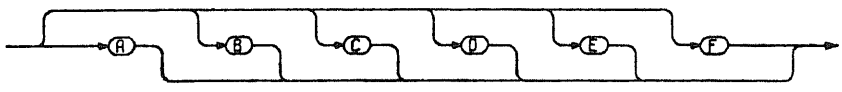
Beispiel:



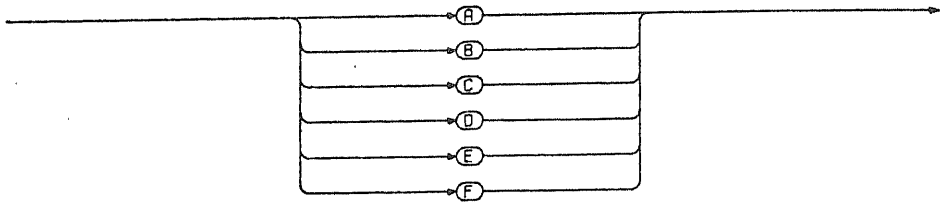
Wir verlangen also von einem Darstellungsprogramm, dass es in der Lage ist, Diagramme auf eine gegebene Seitenbreite einzupassen.

In gewissem Sinne dazu dual ist das Problem, ein hohes, schmales (Unter-) Diagramm so umzuformen, dass der Platz in der Breite ausgenuetzt und dafuer in der Hoehe gespart wird.

Beispiel:



statt



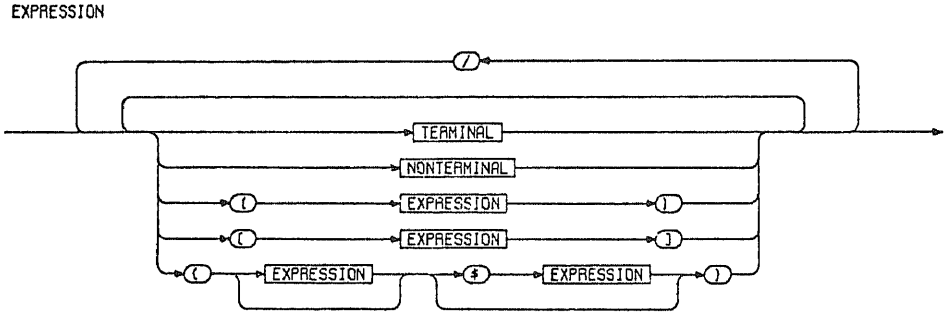
Der Output soll nicht nur moeglichst gut lesbar sein, sondern auch vom Benuetzer nach seinen Wuenschen bequem modifiziert werden koennen. Dafuer sollte er Modifikationsoperationen anwenden koennen, die auf moeglichst hohem Niveau die Beschreibung einer Aenderung erlauben. Das Brechen einer zu langen Serie zum Beispiel sollte durch einen einzigen Befehl erreicht werden, ohne dass muehselig einzelne Koordinaten von Hand veraendert werden muessen.

Diese Modifikationsoperationen sollen auch in dem Sinne sicher sein, als durch ihre Anwendung die Aequivalenz des Diagramms mit dem anfaenglich beschriebenen erhalten bleibt.

1.2 Diverse Eingabeformen

Es werden nun drei Eingabeformen von verschiedener Stufe ins Auge gefasst und, soweit interessant, eingehender behandelt.

Als Extremfall fuer eine Eingabe niederer Stufe wird die Eingabe aller Koordinaten betrachtet. Als Extrem fuer eine Eingabe hoher Stufe betrachten wir die nur sehr wenig Redundanz aufweisende Backus-Naur-Form. Zwischen diesen beiden Extremen liegt die Angabe des abstrakten Diagramms vermittels einer Durchlaufbeschreibung (ohne Koordinaten). Zu Vergleichszwecken wird in allen drei Eingabeformen das folgende Diagramm beschrieben:



A) Eingabe der Geometrie

Wie eine Sprache beschaffen sein soll, in welcher Verbindungen, Koordinaten und Kaestchentexte von Diagrammen beschrieben werden koennen, kann sich der Leser leicht selbst vorstellen. Ohne weitere Erklaerung sei unten das Vergleichsdiagramm in einer solchen Sprache be-

schrieben. Die Details sind nicht von Belang, wichtig ist der optische Gesamteindruck.

```
left(1,2); left(3,4); right(5,6,7,8,9,10,11,12); right(13,14);
right(15,16,17,18); right(19,20,21,22); right(30,31);
right(23,24,25,26,27,28,29); right(32,33); down(2,6);
down(4,7); up(10,3); up(11,1); down(8,13,15,19,23);
up(30,24); down(25,31); down(26,32); up(33,28);
up(29,22,18,14,9);
kaest(TS,./.,1); kaest(NS,/terminal/,9);
kaest(NS,/nonterminal/,14); kaest(TS,/(/,16);
kaest(NS,/expression/,17); kaest(TS,/)/,18);
kaest(TS,/[/,20); kaest(NS,/expression/,21);
kaest(NS,/]/,22); kaest(TS,/{/,24);
kaest(NS,/expression/,25); kaest(TS,/$/,27);
kaest(NS,/expression/,28); kaest(TS,/}/,29);
koord(1,1.5,17.5); koord(2,.,2.3); koord(3,2.2,16.3);
koord(4,.,3.5); koord(5,2.9,0); koord(6,.,2.3);
koord(7,.,3.5); koord(8,.,4); koord(9,.,15.5);
koord(10,.,16.1); koord(11,.,17.3); koord(12,.,20);
koord(13,3.6,.); koord(15,4.3,.); koord(16,.,7.3);
koord(17,.,12.3); koord(19,5,.); koord(20,.,7.3);
koord(21,.,12.3); koord(23,5.7,.); koord(24,.,5.5);
koord(25,.,8.8); koord(26,.,9.3); koord(27,.,10.8);
koord(28,.,14); koord(29,.,15.5); koord(30,6.4,.);
koord(32,6.4,.)    .
```

B) Eingabe der Verbindungen und Kaestchen ohne Koordinaten

(Durchlaufbeschreibung)

Fuer diese zweite Beschreibungsmethode werden die orientierten Kanten des Diagramms durchlaufen, dabei Richtungsaenderungen und ´´Sehenswuerdigkeiten´´ (Kaestchen) vermerkt und Marken gesetzt, falls ein Punkt spaeter aus einer anderen Richtung wieder erreicht werden soll.

Zu dieser Methode werden teilweise Syntax und Semantik einer Sprache angegeben. Da diese Sprache implementiert wurde und einige Erfahrung mit ihrer Benuetzung vorliegt, wird sie etwas ausfuehrlicher behandelt.

Die Beschreibung eines Diagramms besteht aus:

- Deklaration von Symbolen (Kaestcheninhalte)
- Beschreibung der abstrakten Struktur
- Angaben fuer die Nivellierung von Punktreihen und -Spalten.

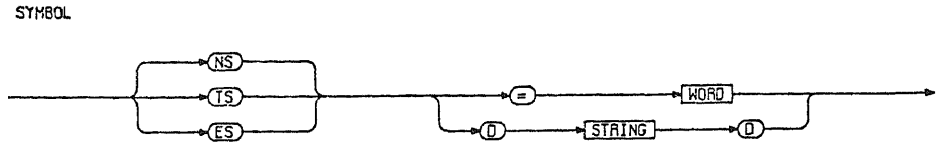
Deklaration von Symbolen

Terminalsymbole, Nichterminalsymbole, usw. koennen in der Strukturbeschreibung explizit eingefuehrt werden oder im Deklarationsteil mit einem Bezeichner identifiziert werden.

Beschreibung der Struktur

Die einfachen Elemente der Strukturbeschreibung sind die vier Bewegungen RIGHT, LEFT, UP, DOWN, die Positionierung POSIT, die Marken M<i> und die Wortsymbole. Die Marken M<i> werden implizit durch ihr erstes Auftreten deklariert.

Wortsymbole genuegen der folgenden Syntax:

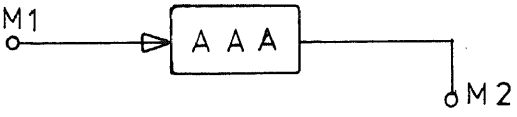


Hierbei steht NS fuer Nichtterminalsymbol, TS fuer Terminalsymbol und ES fuer Fehlersymbol.
Und in der Form NS = ws ist ws ein deklariertes Wortsymbol.
Beispiel: TS = slash : M4 oder NS/label/ : M5

Durch die Positionierungsoperation POSIT(M<i>) wird der Punkt mit der Marke M<i> zum aktuellen Punkt.
Die Operation RIGHT(M<i>) induziert eine Verbindung vom aktuellen Punkt zum Punkt M<i>, wobei der Punkt mit der Marke M<i> aktueller Punkt wird.
Enthaelt RIGHT keinen Markenparameter und folgt auf die Rechtsbewegung eine Marke M<i> oder ein Wortsymbol WS, so wird der Punkt mit der Marke M<i> respektive der Punkt, der durch WS repraesentiert wird, zum aktuellen Punkt.
Folgt auf eine parameterlose Rechtsbewegung jedoch keine Marke und kein Wortsymbol, so wird implizit eine interne Marke fuer den neuen aktuellen Punkt definiert.
Wird ein Wortsymbol durch eine Rechtsbewegung erreicht, so muss es auch mit einer Rechtsbewegung verlassen werden. Diese auslaufende Rechtsbewegung wird implizit generiert.

Beispiel

POSIT(M1); RIGHT; NS/aaa/; DOWN(M2)



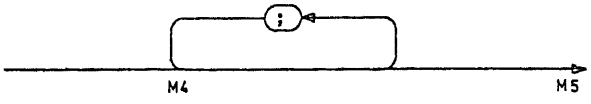
Alles, was fuer Rechtsbewegungen gilt, gilt sinngemaess auch fuer Links-, Auf- und Abbewegungen.

Folgen auf eine Bewegung B mehrere Marken oder Wortsymbole ohne verbindende Bewegungen, so wird die Bewegung B implizit zwischen diesen Elementen generiert.

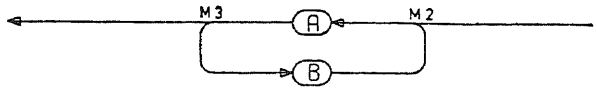
Mit den vier Bewegungen und der Positionierung koennte der Durchlauf durch die Kanten eines Diagramms vollstaendig beschrieben werden. Zur Vereinfachung, bzw. bessern Verstaendlichkeit wurden jedoch drei weitere Sprachelemente eingefuehrt.

Beispiele von LOOPUP und LOOPDOWN:

RIGHT(M4); RIGHT; LOOPUP(M4); TS/;/ ; RIGHT(M5)



```
LEFT(M2); TS/a/; M3; LOOPDOWN(M2) NS/b/; LEFT
```

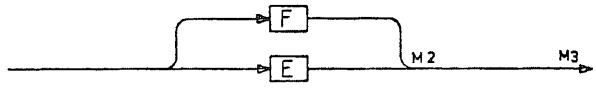


Die Zielmarke des LOOPs muss vor dem LOOP-Aufruf definiert worden sein.

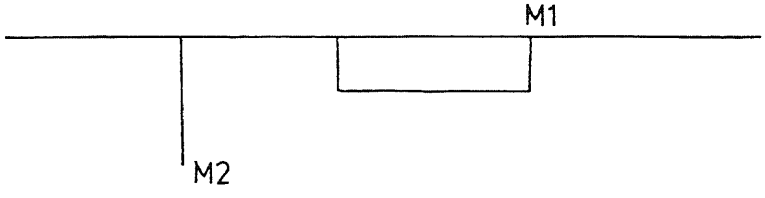
Analog wie LOOPUP und LOOPDOWN funktionieren FORWUP und FORWDOWN.

Beispiel:

```
RIGHT; FORWUP(M2) TS/f/ ; TS/e/; M2; M3
```



Das dritte Sprachelement, welches eine implizite Rueckpositionierung zu einem Ausgangspunkt beinhaltet, ist die BRANCH-Anweisung. Beim Auftreten des Symbols BRANCH wird die interne Marke des aktuellen Punktes festgehalten und beim Auftreten des Symbols END wird eine Rueckpositionierung zu dieser Marke generiert.
Beispiel




```
RIGHT;
BRANCH RIGHT;
       BRANCH DOWN; RIGHT; UP(M1)
       END;
       RIGHT(M1); RIGHT
END; DOWN(M2)
```

Die Nivellierung

Mit der Strukturbeschreibung werden keine Angaben ueber die Niveaux der einzelnen Punkte eingegeben. Wuenscht man z.B. die horizontal nicht verketteten Punkte mit den Marken M2 und M4 auf dasselbe Niveau zu setzen, so bedient man sich der Anweisung SAMEX(M2, M4).

Beschrieben wird das Vergleichsdiagramm EXPRESSION damit so:

```
TITLE/expression/;
SYMBOLS  e = /expression/, t = /terminal/,
         nont = /nonterminal/;
STRUCTURE
   RIGHT(M1); M2; M3;
   BRANCH RIGHT; NS = t; M4; M5;
      BRANCH UP; LEFT; DOWN(M2) END; RIGHT(M6);
```

```
      LOOPUP(M1) TS./.; RIGHT
   END; DOWN;
   BRANCH RIGHT; NS = nont; M7; UP(M4) END; DOWN;
   BRANCH RIGHT; TS/(/; NS = e; TS/)/; M8; UP(M7) END; DOWN;
   BRANCH RIGHT; TS/[/; NS = e; TS/]/; M9; UP(M8) END; DOWN;
   RIGHT; TS/[/; BRANCH DOWN; RIGHT; UP(M1Ø) END; RIGHT;
   NS = e; M1Ø; RIGHT; BRANCH DOWN; RIGHT; UP(M11) END;
   RIGHT; TS/$/; NS = e; M11; TS/]/; UP(M9)
ENDSTRUCT .
```

C) Eingabe der Backus-Naur-Form

Als letzte und interessanteste Eingabeform betrachten wir die Beschreibung der Diagramme durch eine Backus-Naur-Form.
Dabei koennen wir die Backus-Naur-Form verwenden, welche ueblicherweise in der Literatur benuetzt wird, oder eine spezielle, erweiterte Backus- Naur-Form einfuehren, welche fuer das Beschreiben von Diagrammen besonders geeignet ist.
Im ersten Fall haben wir den Vorteil, dass wir ´´Fremdsprachen´´, d.h. Sprachen, die in der Literatur durch Backus-Naur-Form syntaktisch festgelegt sind, ohne weitere Kenntnis der Sprache in Diagramme verwandeln koennen.
Eine Backus-Naur-Form, die nicht im Hinblick auf die Syntaxdiagramme aufgestellt wurde, enthaelt i.A viel mehr syntaktische Klassen, als zu ihrer Beschreibung vernuenftigerweise Diagrammbilder benoetigt werden. In diesem Fall muessen diejenigen Nichtterminale, welche als Diagrammtitel nicht vorzusehen sind, speziell markiert werden. Alle nichtmarkierten Nichtterminale muessen waehrend der Kompilation durch ihre Regel substituiert werden.
Der Hauptnachteil dieses Vorgehens besteht darin, das die so erzeugten Serie-Parallel-Strukturen Diagramme repraesentieren, die nicht eindeutig durchlaufen werden koennen. D.h. es kann Punkte geben, von denen mehr als eine Verzweigung zu identischen Kaestchen fuehrt. Die Schwierigkeiten im Zusammenhang mit solchen Diagrammen werden in [2] erklaert.

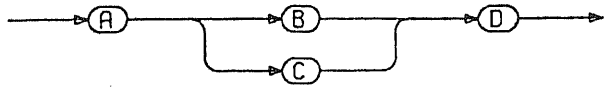
Im Fall der speziellen Backus-Naur-Form haben wir die Moeglichkeit, mit redundanter Information die Diagramme etwas geeigneter zu formen. Wir wollen hier nur diese betrachten.

Syntax der speziellen Backus-Naur-Form

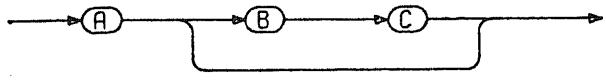
Wie bei der Durchlaufbeschreibung sollen hier auch Kaestcheninhalte durch Bezeichner identifiziert werden koennen. Diese Zuordnung geschieht auf dieselbe Weise wie bei der Durchlaufbeschreibung.

Die eigentliche Backus-Naur-Form besteht aus einer Folge von Produktionen, wobei gegenueber der ueblichen Backus-Naur-Form noch drei Klammeroperationen implementiert sind:

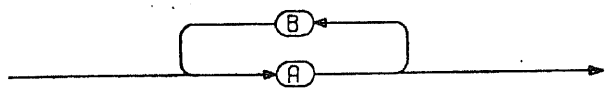
1) Ein Teildiagramm D kann zu weiteren Elementen in serie oder parallel gesetzt werden, wenn die Beschreibung von D in runde Klammern gesetzt wird.
   Beispiel: a (b / c) d
                D



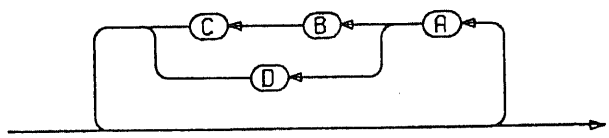
2) Ein Teildiagramm D kann durch eine Leerkante ueberbrueckt werden, wenn die Beschreibung von D in eckige Klammern gesetzt wird.
Beispiel: a [b c]
D



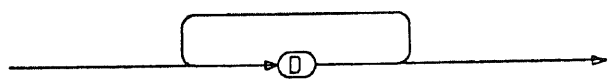
3) Ein Teildiagramm D2 kann ein Teildiagramm D1 ruecklaeufig ueberbruecken, wenn die Beschreibungen von D1 und D2 durch das Zeichen $ getrennt werden, und das Ganze in geschweifte Klammern gesetzt wird.
Beispiel: { a $ b }
D1 D2



Das Teildiagramm D1 darf hier auch leer sein. Man beachte, dass D2 in der Durchlaufreihenfolge beschrieben werden muss.
Beispiel: { $ a (b c / d) }



Ist das Teildiagramm D2 leer, so muss auch das Zeichen $ fehlen.
Beispiel: { d }



Das Diagramm EXPRESSION beschreibt einen Ausdruck in spezieller Backus-Naur-Form und ist auch gerade das Beispiel, welches in den beiden andern Eingabeformen dargestellt wird:

```
BNF
SYM  lrund = /(/, rrund = /)/, leckig = /[/; reckig = /]/,
     lgeschw = /{/, rgeschw = /}/, dollar = /$/, slash = ./. ,
     exp = /expression/;
1 <exp> ::= { { <terminal> / <nonterminal> /
                lrund <exp> rrund /
                leckig <exp> reckig /
                lgeschw [ <exp> ] [ dollar <exp> ] rgeschw }
              $ slash }
Ø
```

Naehere Angaben zu dieser Form der Eingabe koennen [2] entnommen werden.

1.3 Datenstrukturen zu den Eingabeformen

In diesem Abschnitt wollen wir die Datenstrukturen betrachten, welche den verschiedenen Eingabeformen in natuerlicher Weise zugeordnet sind.

Zu jedem beschriebenen Diagramm wird waehrend seiner Kompilation ein Datenobjekt in der zur Beschreibung gehoerenden Datenstruktur aufgebaut. Nach der Kompilation des Diagramms muss dieses Datenobjekt in ein Objekt einer Struktur transformiert werden, welche von der Plot- oder Printplotprozedur leicht verarbeitet werden kann.
Eine solche Struktur muss fuer jeden Diagrammeckpunkt die Koordinaten, seine Nachbarpunkte und den Durchlaufsinn der anstossenden Kanten, sowie Information ueber allfaellig vorhandene Kaestchen auf diesen Kanten enthalten.
Da zu den Angaben ueber einen Punkt mit vorgegebener Nummer direkter Zugriff moeglich sein soll, wird diese Struktur ein Array von Records sein:

```
openstruct =
   array [1..ptemax] of
      record up, down, right, left: 0..ptemax;
         grpo: graphpo; (* Zeiger auf ein Record mit Information
                           ueber ein Kaestchen *)
         level: 0..maxlevel; (* ganzzahlige y-Koordinate fuer
                                 Printplot *)
         breadth: 0..maxbread; (* analoge x-Koordinate *)
         x, y: real; (* reelle Koordinaten fuer Plot *)
         dir: packed array [1..4] of boolean;
                                 (* Orientierung der Kanten *)
         knee: set of 1..12; (* Form der Verzweigung *)
      end
```

Die Aufgabe der Plotprozedur ist es nun, die Information in diesem Array zu vervollstaendigen (knee) und in eine Sequenz von elementaren Plotoperationen zu zerlegen, welche mit moeglichst wenigen Leerbewegungen und Federstatuswechseln durchlaufen werden kann.

Datenstruktur fuer die Eingabe der Geometrie

Bei der geometrischen Eingabe werden Punktnummern, Verbindungen und Koordinaten direkt eingegeben und koennen somit sofort in ein Array vom Typ OPENSTRUCT abgelegt werden.
In diesem Fall besteht das ganze Darstellungsprogramm nur aus einem Kompilations- und einem Plotterteil.

Datenstruktur fuer die Durchlaufbeschreibung

Beim Durchlauf durch das Diagramm werden alle neu angetroffenen Punkte mit einer Nummer versehen und mit den schon durchlaufenen Punk-

ten verbunden. Koordinaten werden noch keine berechnet.
Die Datenstruktur ist also ebenfalls ein Array von Records.
Hier enthalten die Records aber keine Koordinaten wie bei der geometrischen Beschreibung.
Aufgabe des Darstellungsprogramms ist es dann, Koordinaten fuer die Punkte zu finden, die mit den Verbindungen und den Kaestchenlaengen vereinbar sind.

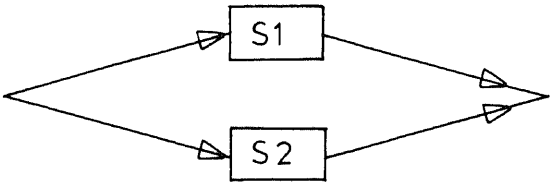
Datenstruktur fuer die Backus-Naur-Form

Die Backus-Naur-Form beschreibt Serien durch Konkatenation von Elementen und Parallelen durch Trennung von Serien durch das Symbol '/'.
Elemente sind hierbei Kaestchen oder komplexere Unterdiagramme.
Als Datenstruktur ergibt sich also in zwangsloser Weise der rekursive Serie-Parallel-Graph.
Ein solcher Serie-Parallel-Graph kann definiert werden durch:

Serie-Parallel-Graph:
<Element> ::= Kaestchen / <Serie-Parallel-Graph>
<Serie> ::= <Element> / <Element> '.' <Serie>
<Serie-Parallel-Graph> ::= <Serie> / <Serie> '/'
<Serie-Parallel-Graph>

Dabei bedeutet '.' das Hintereinanderfuegen von Elementen:

und '/' das Parallelfuehren von Serien:



Diese rekursiven Definitionen koennen folgendermassen durch PASCAL-Typen simuliert werden:
GRAPHEL sei ein Recordtyp, welcher dem Festhalten der Information fuer ein Kaestchen dient (Text und Kaestchenart).
GRAPHPO sei dann der Typ des Zeigers auf ein solches Record.
LINKTYPE sei der Typ, welcher die Information ueber ein Element enthaelt. LINKTYPE kann in PASCAL demzufolge so aussehen:

```
linktype = packed record
             atomic: graphpo;
             nonatom: headpo;
             nextlink: linkpo
           end
```

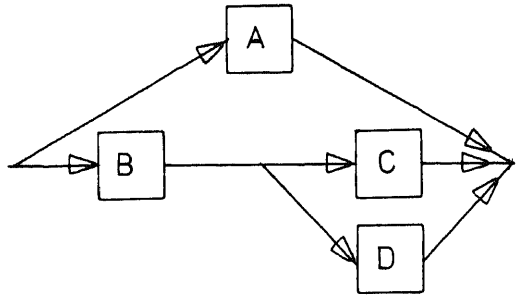
Dabei ist LINKPO der Typ eines Zeigers, welcher auf Records vom Typ LINKTYPE zeigt.

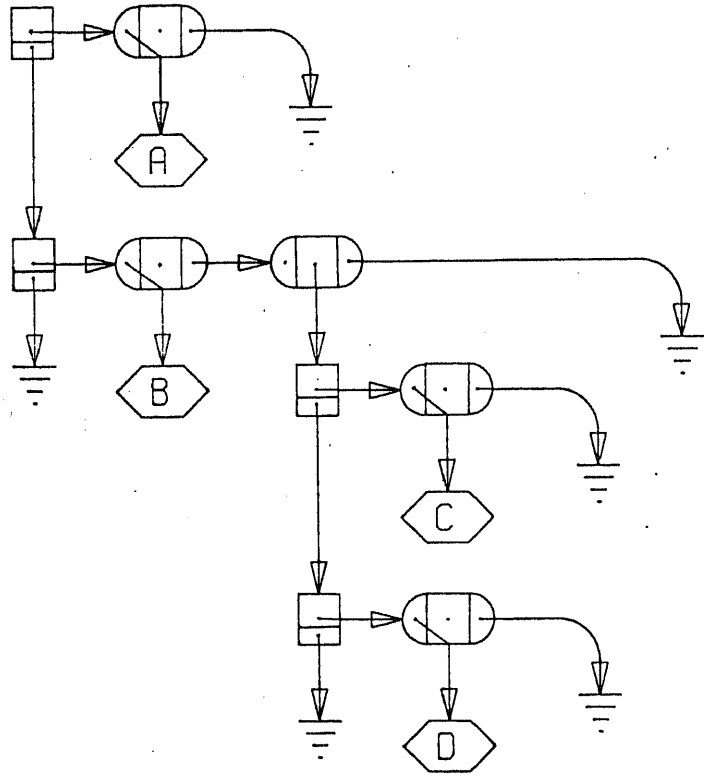
Am Kopf der Serie steht ein Record vom Typ HEADTYPE:

```
headtype = packed record
             firstlink: linkpo; nexthead: headpo
           end
```

HEADPO ist der Typ des Zeigers, welcher auf Records vom Typ HEADTYPE zeigt.
Verschiedene parallele Serien sind ueber ihre Koepfe verbunden.

Beispiel eines Datenobjektes fuer einen Serie-Parallel-Graphen:





1.4 Editionsoperationen

Unter Editionsoperationen wollen wir hier Operationen verstehen, welche nach der Kompilation der Beschreibung eines Diagramms aufgebaut sind.
Solche Operationen koennen mit Vorteil im interaktiven Betrieb auf Datenstrukturen angewandt werden, welche zu starr sind, um direkt alle denkbaren und wuenschbaren Diagramme adaequat auszudruecken.
So werden derartige Operationen kaum auf Diagrammen ausgefuehrt, welche geometrisch eingegeben wurden, da diese Eingabe ja flexibel genug ist, um das gewuenschte Diagramm gleich von Anfang richtig auszudruekken.
Dagegen ergibt die Einschraenkung auf Serie-Parallel-Graphen haeufig Diagramme, welche die aesthetischen Forderungen des Benuetzers und die Anforderungen an die Raumeinteilung auf dem Papier keineswegs erfuellen.
Hier leisten machtvolle Modifikationsoperationen, welche auf ganze Teildiagramme wirken, wertvolle Dienste.

Die Menge der Editionsoperationen, die auf einem Diagramm ausgefuehrt werden kann, wird also durch die Datenstruktur gegeben respektive eingeschraenkt.

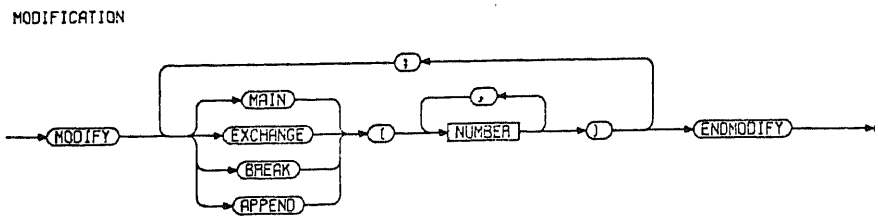
Im Fall der geometrischen Eingabe koennen neue Punkte eingefuehrt oder alte gestrichen werden. Weiter kann man Koordinaten von vorhandenen Punkten veraendern. Dies hat aber meist zur Folge, dass die Koordinaten einer grossen Zahl von weiteren Punkten ebenfalls modifiziert werden muessen.

Bei der Durchlaufbeschreibung koennen neue Punkte eingefuehrt und alte gestrichen werden. Daneben koennen verschiedene Punkte auf dasselbe horizontale oder vertikale Niveau gesetzt werden.

Die Datenstruktur, welche aus der Backus-Naur-Form entsteht, laesst komplexere Operationen zu, die sehr einfach beschrieben werden koennen, so z.B. Vertauschen von Serien, Einsetzen neuer Hauptniveaux, Brechen von Serien, Hintereinanderfuegen von Parallelen, usw.
Bei dieser Eingabeform ist es sinnvoll, sich auf derartige Operationen zu beschraenken, welche ein Diagramm nur auf ein aequivalentes abbilden koennen. Damit erhalten wir die Sicherheit, dass ein Diagramm auch nach Erdulden einer groesseren Zahl von Modifikationen seine urspruengliche Bedeutung noch nicht verloren hat.
Bei der geometrischen- oder der Durchlaufbeschreibung kann diese Sicherheit nicht so leicht erreicht werden, da schon die Umstaendlichkeit der urspruenglichen Beschreibung genuegend Moeglichkeiten eroeffnet aufgrund eines Fehlers ein anderes als das intendierte Diagramm zu generieren.

Die wichtigsten Operationen auf der Serie-Parallel-Struktur seien hier angefuehrt:

Die Syntax der Befehle genuegt dem Diagramm:

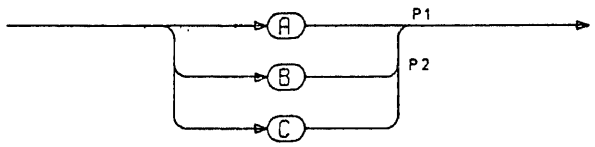


Parameter der Prozeduren MAIN, EXCHANGE, BREAK und APPEND sind die internen Punktnummern des Diagramms. (Im Stapelbetrieb muessen diese Nummern in einem ersten Programmlauf sichtbar gemacht werden. Z.B. durch Ausdrucken im Prinplot).

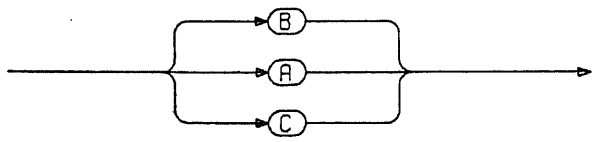
Die Prozedur EXCHANGE hat genau zwei Parameter. Die Prozeduren MAIN, BREAK und APPEND je mindestens einen.

EXCHANGE(p1,p2) bewirkt das Vertauschen der beiden Parallelen, auf denen p1 respektive p2 liegen. Diejenige Parallele, welche vor dem Vertauschen auf dem Hauptniveau lag, liegt nachher weiterhin auf dem Hauptniveau.

Beispiel:

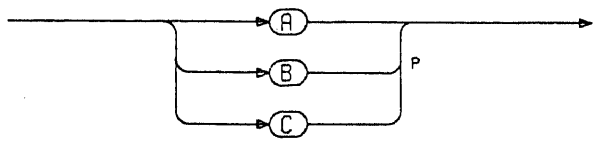


EXCHANGE(p1,p2)

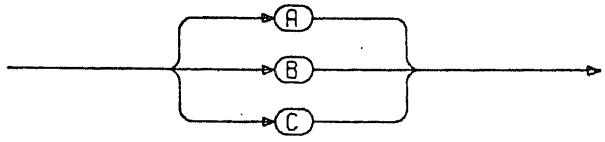


MAIN(p) bewirkt, dass die Serie mit dem Punkt p auf das Hauptniveau zu liegen kommt.

Beispiel:

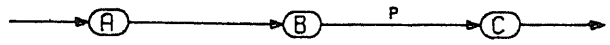


MAIN(p)

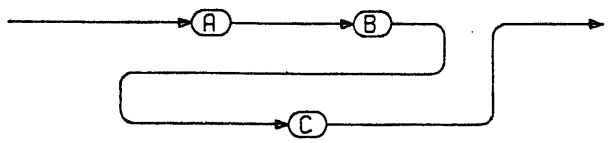


BREAK(p) bewirkt, dass die Serie, welche p enthaelt, bei p gebrochen und zum vorangehenden Serieteil parallel gefuehrt wird.

Beispiel:

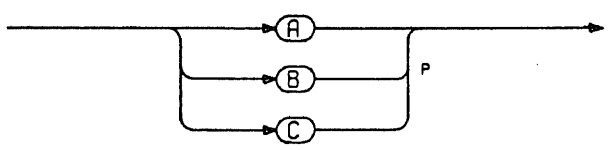


BREAK(p)

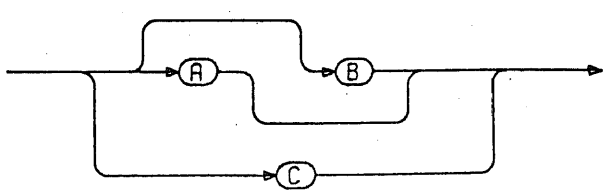


APPEND(p) bewirkt, dass die Parallele, auf welcher p liegt, zu den darueber liegenden Parallelen in Serie gefuehrt wird.

Beispiel:



APPEND(p)



BEFEHL(p1,...,pk) ist gleichwertig wie BEFEHL(p1),...,BEFEHL(pk), fuer BEFEHL gleich MAIN, BREAK oder APPEND.

2. Implementierung von Zeichenprogrammen

2.1 Implementationsprobleme bei der Durchlaufbeschreibung

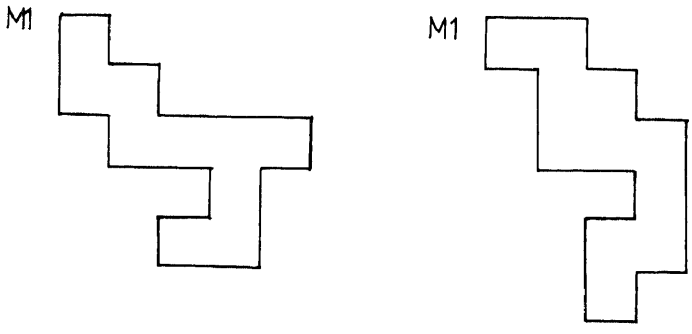
Obwohl wir der Diagrammbeschreibung durch Backus-Naur-Form gegenueber der Durchlaufbeschreibung eindeutig den Vorzug geben, soll in diesem Abschnitt kurz auf die Implementationsprobleme bei letzterer Methode eingegangen werden.
Dies hat seinen Grund u.a. darin, dass eine Version der Duchlaufbeschreibung spaeter zur Charakterisierung von Substrukturen in der Serie-Parallel-Struktur verwendet wird.

Das Resultat der Kompilation einer Diagrammbeschreibung ist ein abstraktes Diagramm. Ein solches abstraktes Diagramm muss durch Angabe von Koordinaten zu einem Diagramm erweitert werden, sofern dies moeglich ist. Andernfalls sollte ein entsprechender Vermerk ausgegeben werden.

Betrachten wir etwa die folgende Beschreibung einer geschlossenen Kurve in der Ebene, so sehen wir, dass damit gar nicht eindeutig eine geometrische Figur charakterisiert ist:

```
M1; down; right; down; right; down; left; down; right; up;
    right; up; left; up; left; up; left(M1)
```

Dieses wird z.B. durch die nachstehenden beiden Figuren erfuellt:



Will man das Problem trotzdem einigermassen zufriedenstellend loesen, so kann man davon ausgehen, dass bei Syntaxdiagrammen i.A. nicht allzuviele Mehrdeutigkeiten der Durchlaufbeschreibung auftreten werden.
Dabei kann man voraussetzen, dass die Nivellierung meist eindeutig bestimmt werden kann, wenn man die Transitivitaet der Ordnungsrelation auf den Niveaux und die Information ueber die Schachtelung der durch die vertikalen Kanten aus den Horizontalen geschnittenen Intervalle beruecksichtigt.

Diese Information reicht in den meisten, aber doch nicht in allen Faellen aus, um die Nivellierung der Figur zu rekonstruieren.
Dabei wurde zudem das Problem der Nivellierung von demjenigen der Breiteneinteilung getrennt, was u.U. auch zu einer falschen Einbettung fuehren kann.

Als zweites Problem neben der Nivellierung stellt sich wie eben erwaehnt das Problem der Breiteneinteilung.
Fuer eine einfache Loesung dieser Aufgabe setzen wir voraus, dass jeder Punkt des Diagramms vom Eingangspunkt am linken Rand aus durch Rechts-, Auf- und Abbewegungen erreicht werden kann.
Ist dann ein nivelliertes Diagramm vorgegeben, ist es nicht mehr schwierig, einen Satz von Hoehen-Koordinaten fuer das Diagramm zu bestimmen, falls ueberhaupt einer existiert, welcher mit der Nivellierung und den Kaestchenbreiten vertraeglich ist.
Einen solchen Koordinatensatz erhalten wir etwa, indem wir die ´´linksbuendige´´ (oder rechtsbuendige) Einbettung angeben, welche mit obiger Voraussetzung ueber die Erreichbarkeit der Punkte unschwer zu berechnen ist.
Der benoetigte Algorithmus entspricht einem einfachen Terminplanungsverfahren der Netzplantechnik, wenn man die horizontalen Kanten als verschiedene Arbeiten und die Kaestchenlaengen als die zugehoerigen Arbeitszeiten auffasst.
Diese Einbettung ist jedoch wegen der Konzentration der Figurelemente am Rand und der Dichte dieser Elemente aus aesthetischen und praktischen Gruenden wenig brauchbar.
Durch Mittelung der Koordinaten der links- und rechtsbuendigen Einbettung koennen zwar die Bildraender entlastet und ein etwas symmetrischeres Aussehen erreicht werden. Eine optimale Nutzung der Papierbreite findet aber auch bei dieser zentrierten Darstellung nicht statt.
Versucht man durch lokalen Ausgleich einige Elemente auf einem Niveau zu verteilen, so kann dies zu Platzschwierigkeiten auf anderen Niveaux fuehren, welche mit diesem Niveau durch vertikale Kanten verbunden sind.

Ein globales Einteilungsverfahren kann man erhalten, indem man einen Weg maximaler Laenge von links nach rechts durch das Diagramm sucht, und den freien Platz entlang dieses Weges geeignet verteilt. (Vertikale Kanten sollen dazu mit der Laenge Null in die Rechnung eingehen).
Fuer den naechsten Schritt muessen die Koordinaten der Punkte auf dem maximalen Weg festgehalten und ein weiterer maximaler Teilweg im Diagramm bestimmt werden, usf.
Konkreter kann man sagen, dass in jedem Schritt ein Weg maximaler Laenge von einem Anfangspunkt zu einem Endpunkt gesucht wird, wobei die Koordinaten einiger Punkte bereits festgelegt sind (nenne diese Punkte feste Punkte).
Die Gesamtheit der Anfangspunkte sei diejenige Menge von festen Punkten, welche vom Eingang des Diagramms mit Rechts-, Auf- und Abbewegungen nur ueber feste Punkte erreicht werden koennen.
Die Gesamtheit der Endpunkte sei analog definiert.
Zu Beginn des Verfahrens seien Ein- und Ausgangspunkt des Diagramms feste Punkte. Die Laenge eines Weges ist immer zwischen diesen beiden Punkten zu messen.
Es werden so lange maximale Restwege bestimmt, bis alle Punkte fest sind.
Dieses Verfahren kann dazu dienen, den Begriff der ausgeglichenen Einbettung zu definieren, wenn man noch festlegt, was unter der ´´geeigneten Verteilung´´ des freien Raumes zu verstehen ist.
Leider funktioniert das Verfahren nur, wenn man den Freiraum auf die einzelnen Kanten proportional zur minimalen Kantenlaenge verteilt.
Eine andere Verteilstrategie, z.B. gleicher Anteil fuer alle Kanten, wird i.A. zu unvertraeglichen Koordinatenkonfigurationen fuehren.
Die Einschraenkung auf das proportionale Verteilschema ergibt fuer Se-

rie-Parallel-Graphen nicht unbedingt eine optimale Loesung.

Zusammenfassend muss man festhalten, dass die Information, welche uns die Durchlaufbeschreibung von einem Diagramm bereitstellt, zu dessen Rekonstruktion nicht ganz ausreicht.
Insbesondere duerfte es schwierig oder gar unmoeglich sein, ein Programm zu schreiben, welches die Nivellierung in jedem Fall korrekt ausfuehren kann.
Die Information koennte dadurch vervollstaendigt werden, dass man jeden Up- oder Down-Befehl nur ein Niveau ueberbruecken laesst. Gegebenenfalls koennte diesen beiden Befehlen ein Repetitionsparameter beigefuegt werden.
Mit diesem Zusatz laesst sich das Nivellierungsproblem ohne jede Muehe loesen. Unbeantwortet bleibt dann allerdings noch die Frage nach einer aesthetisch befriedigenden Breiteneinteilung.

So ergibt sich also vom Standpunkt der Implementierung ein schwerwiegendes Argument gegen die Beschreibung der Diagramme mittels unserer Durchlaufbeschreibung.

Um sich ein Bild ueber die Moeglichkeiten des Programms zu machen, welches die Durchlaufbeschreibung verarbeitet, betrachte man die Syntaxdiagramme in den Dissertationen von U. Ammann [1], H.-P. Frei [4], E. Marmier [5] und H. Sandmayr [6].

2.2 Serie-Parallel-Graphen als Diagramme

Da die rekursive Definition der Diagramme durch die Backus-Naur-Form direkt derjenigen der Serie-Parallel-Graphen entspricht, kann waehrend des Einlesens der Beschreibung sofort die Serie-Parallel-Struktur mittels rekursiver Prozeduren aufgebaut werden.
Die so erzeugte Information haelt fest, welche Elemente in serie und welche parallel gesetzt werden.
Ueber Koordinaten von Diagrammeckpunkten oder ueber diese Punkte selbst ist zu diesem Zeitpunkt noch nichts weiter bekannt.
Es gilt jetzt also die Serie-Parallel-Struktur zu interpretieren, um dadurch zu einer Datendarstellung zu gelangen, welche vom Plotprogramm verarbeitet werden kann.
Hierfuer muessen die noetigen Diagrammeckpunkte eingefuehrt und miteinander verbunden werden.
Daneben kann man eine Breiteneinteilung und eine Nivellierung fuer den Graphen bestimmen.
Fuer diese drei einfachen Aufgaben sollen die zugehoerigen Prozeduren kurz vorgefuehrt werden. An ihnen wird deutlich, wie unsere rekursive Datenstruktur durchlaufen wird und wie einfach sich hier die Probleme loesen lassen, die uns bei der Durchlaufbeschreibung zu schaffen machten.

Der Einfachheit halber nehmen wir in diesem Abschnitt an, dass die verschiedenen Serieelemente immer bei den obersten Parallelen verbunden sein sollen.

Um eine ausgeglichene Breiteneinteilung zu finden, berechnen wir vorerst fuer jedes Serieelement dessen minimal moegliche Breite.

Danach verteilen wir den freien Raum fuer die einzelnen Parallelen unter die Serieelemente dieser Parallelen. Hierbei ist der freie Raum fuer das ganze Diagramm die Differenz aus verfuegbarer Papierbreite und minimal moeglicher Diagrammbreite.

In der folgenden Prozedur bezeichnet DIST den Minimalabstand eines Elements von einem Diagrammeckpunkt. Der Resultatparameter LENGTH wird nach der Ausfuehrung der Prozedur die Minimalbreite desjenigen Elements enthalten, auf welches der Zeiger HEAD zeigt.
Die Minimalbreite fuer jedes komplexe Element (ausser fuer das ganze Diagramm selbst) wird im zugehoerigen Record vom Typ LINKTYPE in der Variablen MINLENGTH abgespeichert.

```
procedure minimallength(head: headpo; var length: integer);
var serieleng, leng: integer; link: linkpo;
begin length := 0;
  while head <> nil do with head^ do
    begin link := firstlink; serieleng := 0;
      while link <> nil do with link^ do
        begin
          if nonatom <> nil then
               minimallength(nonatom,leng) else
          if atomic <> nil then with atomic^ do
            leng := ''length of symbol'' + 2*dist
          else leng := dist;
          minlength := leng; serieleng := serieleng + leng;
          link := nextlink
        end;
      if serieleng < length then length := serieleng;
      head := nexthead
    end
end (* minimallength *)
```

Die Funktionsweise der Prozedur zur Berechnung der Niveaux ist analog zu derjenigen der Prozedur MINIMALLENGTH.
Einen aehnlichen Aufbau weist auch die Prozedur zur Verteilung des freien Raumes auf. Der Parameter FRAME gibt die Gesamtbreite fuer ein Element an:

```
procedure distribute(head: headpo; frame: integer);
var free, linkcount, serielength: integer; link: linkpo;
begin
  while head <> nil do with head^ do
    begin link := firstlink; serielength := 0; linkcount := 0;
      while link <> nil do with link^ do
        begin linkcount := linkcount+1;
          serielength := serielength+minlength;
          link := nextlink
        end;
      free := (frame-serielength) div linkcount;
      link := firstlink;
      while link^.nextlink <> nil do with link^ do
        begin link := nextlink; minlength := minlength+free;
          if nonatom <> nil then distribute(nonatom,minlength)
        end;
      with link^ do
        begin
```

```
          minlength := minlength +
                       free*(frame-serielength) mod linkcount;
          if nonatom <> nil then distribute(nonatom,minlength)
        end;
      head := nexthead
    end
end (* distribute *)
```

Die folgende Prozedur ASSIGNPOINTS ''generiert'' die Nummern der Diagrammeckpunkte und speichert sie vorlaeufig noch in die Records vom Typ LINKTYPE bzw. HEADTYPE ab.
Dazu bezeichnet LINKPOINT das entsprechende Element im Record LINKTYPE und analog HEADPOINT fuer HEADTYPE.
PTNR ist die Anzahl der bisher eingefuehrten Punkte.
Die Parameter BEGPT und ENDPT sind die Nummern des Anfangs- bzw. Endpunkts des aktuellen Serieelements.

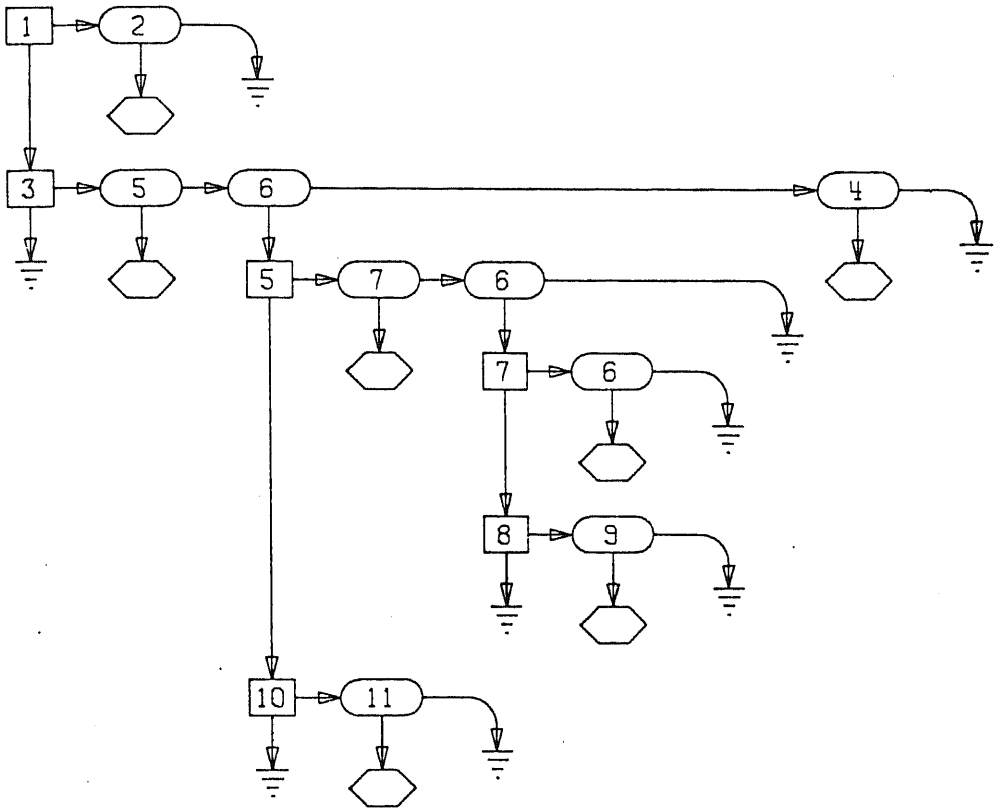
```
procedure assignpoints
            (potohead: headpo; begpt, endpt: pointrange);
var head: headpo; link: linkpo;
    frompoint, topoint: pointrange;
begin head := potohead;
  while head <> nil do with head^ do
    begin
      if head = potohead then frompoint := begpt else
      begin ptnr := ptnr+1; frompoint := ptnr end;
      link := firstlink; headpoint := frompoint;
      while link <> nil do with link^ do
        begin
          if (nextlink = nil) and (head = potohead) then
            topoint := endpt  else
          begin ptnr := ptnr+1; topoint := ptnr end;
          linkpoint := topoint;
          if nonatom <> nil then
            assignpoints(nonatom,frompoint,topoint);
          frompoint := topoint; link := nextlink;
        end;
      head := nexthead
    end
end (* assignpoints *)
```

In derselben Prozedur haette man auch gleich die Punkte vertikal und horizontal verbinden und ihre Orientierung festhalten koennen. Darueberhinaus haette man den einzelnen Punkten auch gleich die Breitenkoordinate zuweisen koennen. Diese Koordinate erhaelt man durch aufsummieren der Groessen MINLENGTH, die mit der Prozedur DISTRIBUTE berechnet wurden.
Die resultierenden Informationen stuenden dann in einem Array vom Typ PLOTSTRUCT und koennten sofort dem Plotprogramm zugefuehrt werden.
Auf eine Liste der entsprechenden Programmteile wurde der Uebersichtlichkeit halber verzichtet.

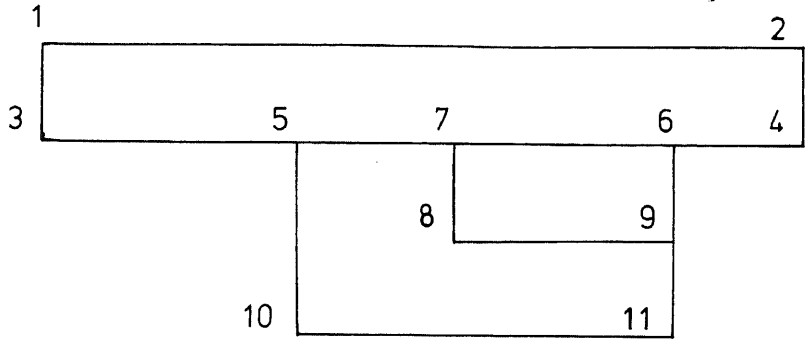
Initialisiert und aufgerufen wird die Prozedur durch

```
ptnr := 2; assignpoints(pointer, 1, 2)
```

Als Beispiel fuer eine Nummernzuteilung durch obige Prozedur betrachten wir folgendes Datenschema mit eingesetzten LINKPOINT und HEADPOINT:

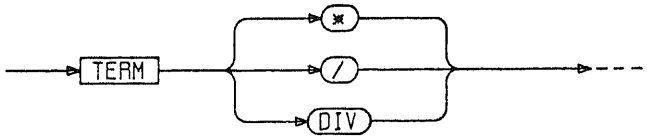


Diese Punkte werden zu folgendem schematischen Diagramm verbunden:



Das Problem eine Regelbeschreibung in spezieller Backus-Naur-Form in eine Diagrammzeichnung umzusetzen ist nun mit den oben aufgezeichneten oder erwaehnten einfachen Prozeduren zusammen mit der Kompilations- und Plotprozedur eigentlich geloest.
Allerdings ist nicht zu uebersehen, dass die resultierenden Diagramme einem recht starren Formenprinzip unterworfen sind. So ist es z.B. nicht einmal moeglich, das folgende Teildiagramm zu erzeugen, weil

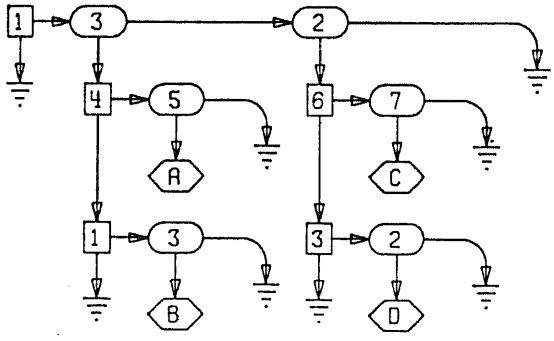
dort die Serieelemente nicht bei den obersten Parallelen verbunden sind:



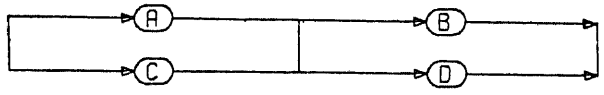
Selbstverstaendlich ist es nicht besonders schwierig, die Prozeduren derart zu erweitern, dass die Verbindung von Serieelementen bei jeder Parallelen moeglich ist.
Genauso koennte ausgehend von den obigen Prozeduren irgend eine andere einfache Bildverfeinerung ohne grosse Muehe implementiert werden. Die Schwierigkeiten beginnen sich aber zu melden, sobald man mehrere solcher zusaetzlicher Darstellungsmoeglichkeiten gleichzeitig in einem Programm vereinen will.
Beispiele fuer solche Schwierigkeiten werden in einem spaeteren Abschnitt weiter verfolgt.

2.3 Die primaere Interpretation

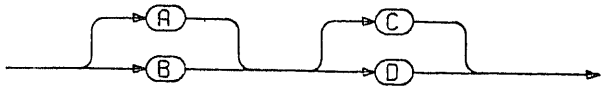
Die erste Interpretation, welche von einer Serie-Parallel-Struktur zu einem abstrakten Diagramm fuehrt, haben wir im Abschnitt 2.2 betrachtet. Dort wurden vermittels der Prozedur ASSIGNPOINTS die Diagrammeckpunkte eingefuehrt.
Diese Prozedur erzeugt aber ausgehend von der urspruenglichen Serie-Parallel-Struktur zuwenig Punkte. Z.B. wird aus der Struktur



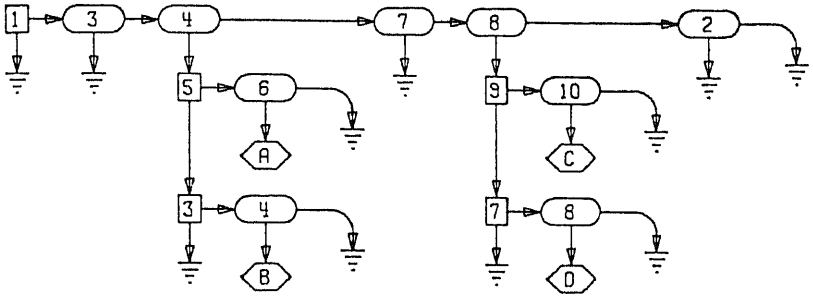
das Diagramm



zusammengehaengt.
Das erhoffte Diagramm

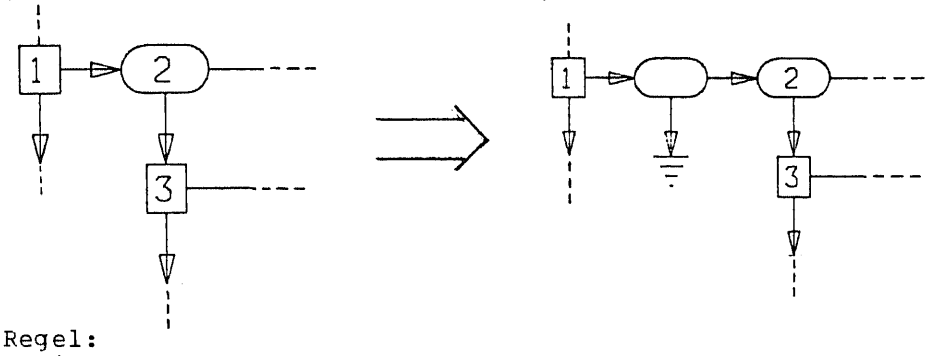


hingegen entsteht aus der Struktur

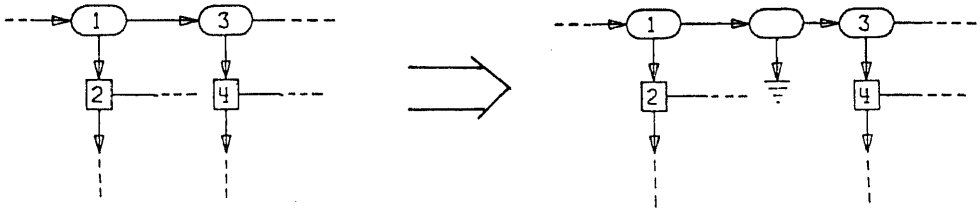


Es muessen demnach fuer die Leerkanten an verschiedenen Stellen in die Serie-Parallel-Struktur ´´Leersymbole´´ eingefuegt werden. Diese Stellen werden mit den folgenden fuenf Regeln erfasst:

Erste Regel:
Setze ein Leersymbol zwischen den Seriekopf und das erste Serieelement, falls dieses nicht atomisch ist.

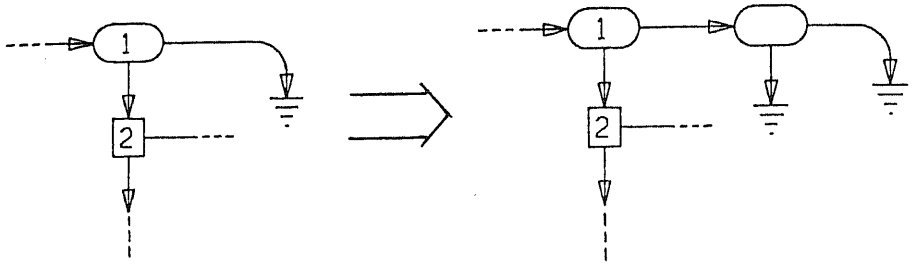


Zweite Regel:
Setze ein Leersymbol zwischen zwei Serieelemente, falls beide nicht atomisch sind.

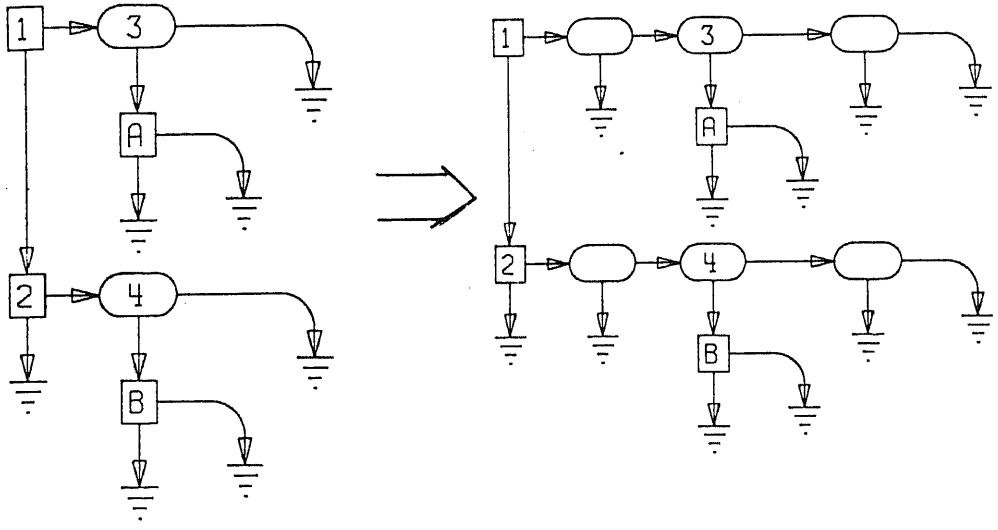


Dritte Regel:
Setze ein Leersymbol ans Ende einer Serie, falls das letzte Element

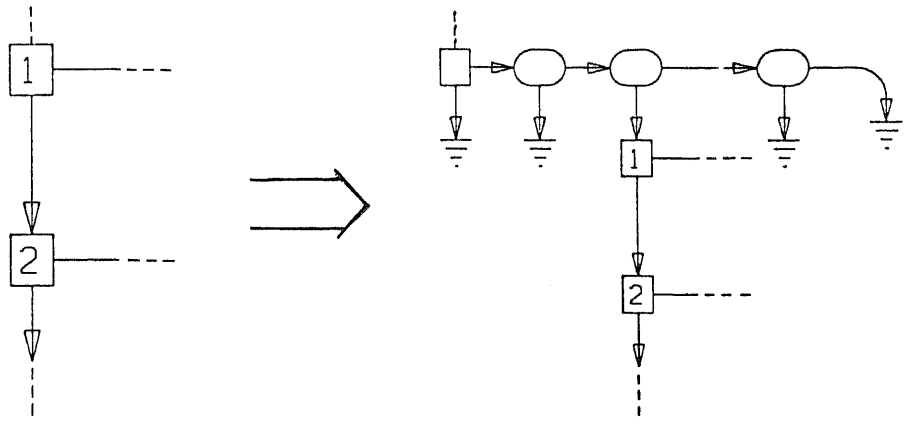
der Serie nicht atomisch ist.



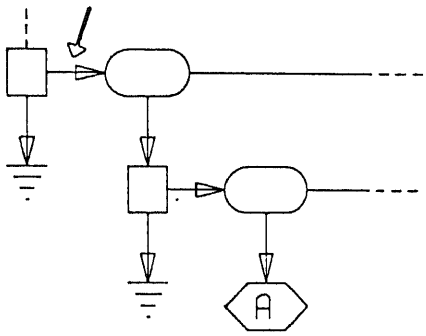
Vierte Regel:
Rahme das untere (obere) Element einer Schleife mit zwei Leersymbolen ein, falls es nicht atomisch ist.



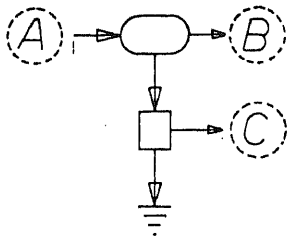
Fuenfte Regel:
Falls das Diagramm auf der ersten Stufe aus mehr als einer parallelen Serie besteht, muss fuer eine Eingangs- und eine Ausgangskante gesorgt werden durch die Erweiterung:



Somit haben wir fuer die primaere Interpretation genuegend Leersymbole eingefuehrt und koennen uns gleich die Frage stellen, ob wir jetzt nicht mit ueberfluessigen Leersymbolen das Diagramm unnoetig komplizieren.
Dies kann i.A. durchaus der Fall sei. Z.B. brauchte man die erste Regel nicht anzuwenden, wenn die Serie mit einem atomischen Symbol A beginnt, welches wie folgt in der Struktur versteckt ist:



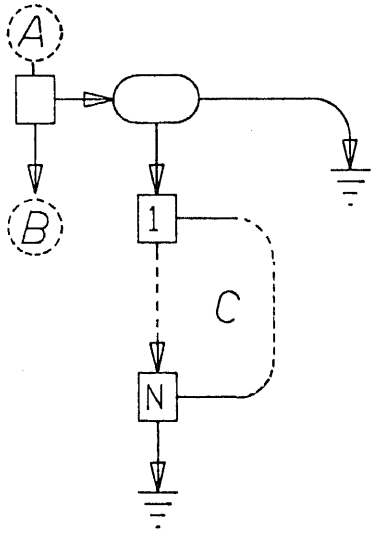
Eine solche Struktur kann durch unnoetigen Gebrauch von runden Klammern (,) in der Eingabe auch tatsaechlich erzeugt werden. Vor allem aber die Modifikationsoperationen, welche spaeter beschrieben werden, koennen zu derartigen Strukturen fuehren.
(Der Pfeil gibt an, wo das ueberfluessige Leersymbol nach der ersten Regel eingefuehrt wird).
Um diesen und einen weiteren Fall auszuschalten (welche uns auch in einem spaeteren Zusammenhang unangenehm auffallen wuerden), fuehren wir die Serie-Parallel-Struktur in eine Normalform ueber.
Eine Serie-Parallel-Struktur in Normalform erfuellt folgende zwei Bedingungen:
1) Kein echtes Unterdiagramm besteht aus nur einer Parallelen.
   Das heisst, die Situation



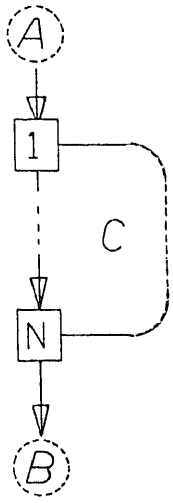
wird uebergefuehrt in die Situation

und dual dazu

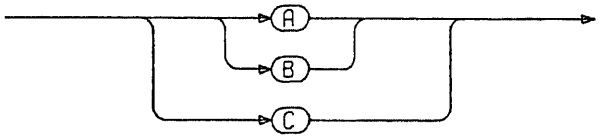
2) Keine Serie besteht aus nur einem nichtatomischen Serieelement ohne rueckfuehrende Parallele *).
Das heisst, die Situation



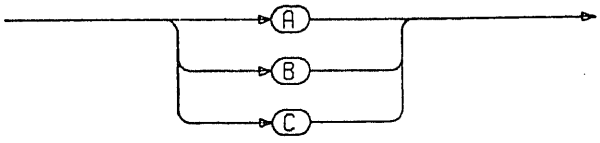
wird uebergefuehrt in die Situation



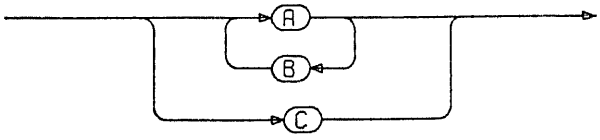
Die Ueberfuehrung in die Normalform hat also zur Folge, dass ueberfluessige runde Klammern auf das Endprodukt keinen Einfluss haben. Als Nachteil dieser Normalisierung ist jedoch zu vermerken, dass der Benuetzer dadurch eine Moeglichkeit verliert, mit der er seine Ausgabe haette steuern koennen.
So bewirkt z.B. die Eingabe (a / b) / c nicht das erhoffte Diagramm



sondern das Diagramm

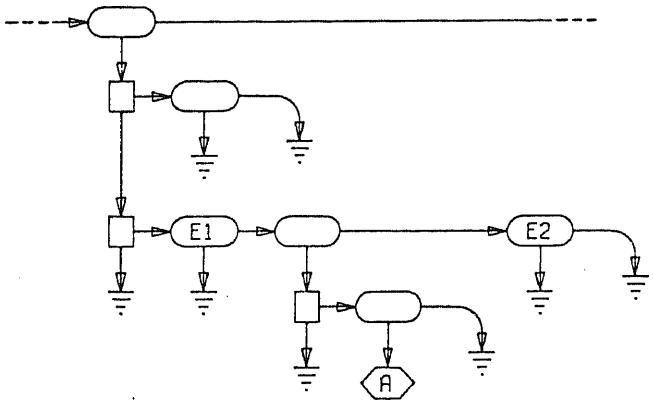


welches auch durch a / b / c beschrieben wird.
Anhand dieser Beispiele erkennt man auch, dass durch die zweite Vereinfachungsregel ohne den Zusatz *) ein falsches Diagramm entsteht, falls das einzige Serieelement auf oberster Stufe eine Schleife darstellt. Es wuerde dann naemlich das Diagramm

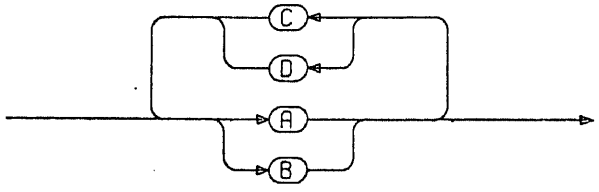


beschrieben durch { a $ b } / c ebenfalls wie dasjenige fuer a / b / c dargestellt.
Bei der Vereinfachung nach 2) muss also noch auf rueckfuehrende Parallelen geprueft werden.

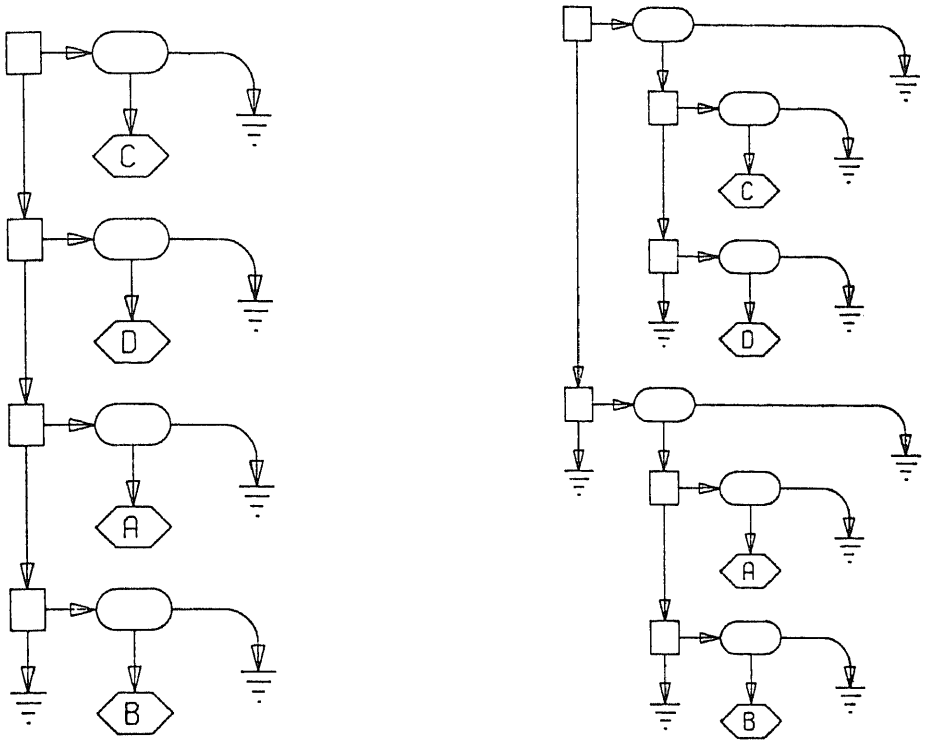
Waehrend die Normalisierung erst nach der Kompilation (ev. Modifikation) stattfindet und die Leersymbole gemaess den Regeln eins bis drei erst danach eingefuegt werden, setzt man die Leersymbole nach Regel vier bereits im Laufe der Kompilation ein. Auf diese Weise koennen in Schleifen ueberfluessige Leersymbole auftauchen, die eigens wieder entfernt werden muessen. Derartige Leersymbole entstehen z.B. durch die Beschreibung { (a) } . In der (unnormalisierten) Struktur sind sie mit e1 und e2 bezeichnet:



Scheinbar koennte man auf diese vierte Regel verzichten, da ja die ersten drei Regel schon fuer Leersymbole in den Schleifen sorgen wuerden, wo dort solche noetig sind. Es ist aber wiederum die Vereinfachung nach 2), welche in jenem Fall falsche Diagramme erzeugen koennte. Z.B. wuerde das Diagramm



mit Struktur (ohne Leersymbole): normalisiert zu:



was offensichtlich nicht korrekt ist.

Im Folgenden werden in Strukturschemata die Leersymbole nicht eingezeichnet, wenn dies der Uebersichtlichkeit dient.

2.4 Interpretationen der Serie-Parallel-Struktur

Im Abschnitt 2.2 haben wir bereits eine einfache Interpretation der Serie-Parallel-Struktur als Diagramm angetroffen. Dort haben wir auch festgestellt, dass eine variablere Interpretationsmoeglichkeit aufgrund zusaetzlicher Information (Z.B. Auszeichnen von beliebigen Hauptniveaux) sehr wuenschbar waere.
Ein paar der einfachsten und nuetzlichsten Operationen auf der Serie-Parallel-Struktur und deren Interpretation sollen hier besprochen werden.

Besonders einfach ist die Vertauschung zweier paralleler Unterdiagramme. Diese kann durch Umhaengen von Zeigern in der Serie-Parallel-Struktur bewirkt werden und benoetigt nicht einmal zusaetzliche Information zur Interpretation.

Eine weitere sehr einfache Operation ist das Festlegen eines Hauptniveau in einer Menge von parallelen Serien.

Die Information darueber, ob eine Serie auf dem Hauptniveau liegt oder nicht, wird in einer logischen Variablen im Kopf der Serie abgelagert, also in einem erweiterten Record vom Typ HEADTYPE.
Die entsprechende logische Variable soll MAINHEAD heissen.
Dadurch, dass Teildiagramme jetzt nach unten und oben ueber ihre horizontale Verbindungslinie hinausragen koennen, kann die Nivellierung nicht mehr in einem rekursiven Durchgang bestimmt werden. Vielmehr muessen in einem ersten Durchgang relative Niveauxabstaende bezueglich der Hauptniveaux errechnet werden. Aus diesen sind dann die absoluten Niveaux geeignet aufzusummieren.
Wie dies konkret bewerkstelligt wird, zeigen die folgenden beiden Prozeduren RELMINMAX und LEVELS.
Zu diesem Zweck musste das Record vom Typ HEADTYPE um eine Groesse RELLEV erweitert werden, welche das relative Niveau bezueglich des Hauptniveau festhaelt. Daneben enthaelt dieses Record in der Groesse HEADPOINT auch die Nummer des Diagrammeckpunktes, welche durch die Prozedur ASSIGNPOINTS generiert wurde. Die entsprechende Groesse im Record vom Typ LINKTYPE lautet LINKPOINT.
Die Resultatparameter RELMAX und RELMIN in der Prozedur RELMINMAX geben also Hoehe und Tiefe an, um welche das durch POTOHEAD referierte Teildiagramm ueber das zugehoerige Hauptniveau hinausragt.
Die aufsummierten Hoehenkoordinaten werden direkt im Array GRAPH vom Typ OPENSTRUCT abgelegt.

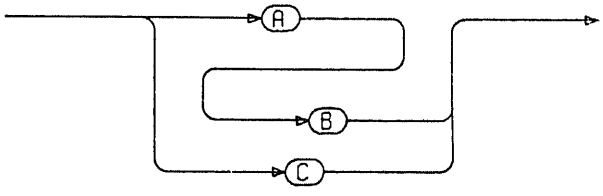
```
procedure relminmax(potohead:headpo;var relmax,relmin:integer);
var head: headpo; link: linkpo;
    l1, l2, max, seriemin, seriemax: integer;
begin head := potohead; max := 0;
  while head <> nil do with head^ do
    begin link := firstlink;
      seriemin := 0; seriemax := 0;
      while link <> nil do with link^ do
        begin
          if  nonatom <> nil then relminmax(nonatom,l1,l2) else
          begin l1 := 1; l2 := 0 end;
          if seriemax < l1 then seriemax := l1;
          if seriemin > l2 then seriemin := l2;
          link := nextlink
        end;
      max := max + seriemax;
      if mainhead then begin relmax := max; max := 0 end;
      rellev := - max;
      max := max - seriemin;
      head := nexthead;
    end;
  relmin := - max;
  head := potohead;
  while not head^.mainhead do with head^ do
    begin rellev := relmax + rellev; head := nexthead
    end
end (* relminmax *)


procedure levels(head: headpo; mainlevel: integer);
var link: linkpo; mainlev: integer;
begin
  while head <> nil do with head^ do
    begin mainlev := rellev + mainlevel;
      graph[headpoint].level := mainlev;
      link := firstlink;
```

```
      while link <> nil do with link^ do
        begin
          if nonatom <> nil then levels(nonatom,mainlev)
          else graph[linkpoint].level := mainlev;
          link := nextlink
        end;
      head := nexthead
    end
end (* levels *)
```

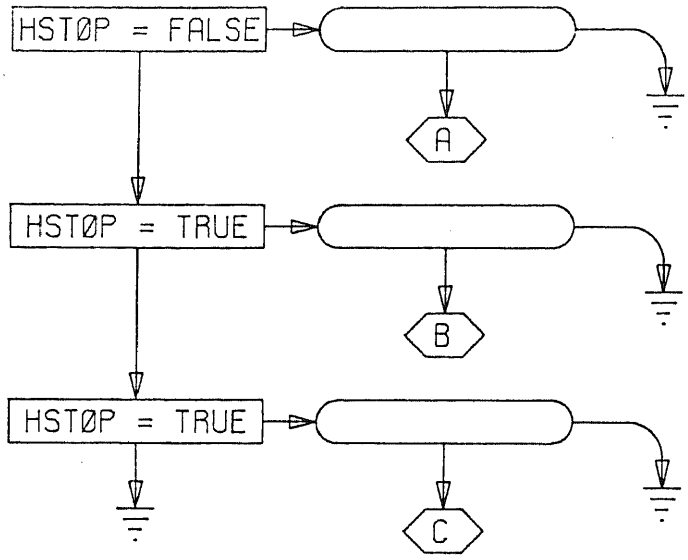
Etwas nuetzlicher, aber auch etwas komplizierter zu realisieren sind die Operationen, welche eine gute Ausnuetzung des Zeichenpapiers ermoeglichen, wie das Brechen zu langer Serien und das Hintereinanderfuegen vieler kurzer Parallelen.

Bezueglich der Hoehen- und Breiteneinteilung und Platzbedarf verhaelt sich eine gebrochene Serie wie eine Folge von Parallelen.
Analog verhalten sich hintereinandergesetzte Parallelen wie eine Serie.
Um die Vorteile der Serie-Parallel-Struktur bei der Breiten- und Hoeheneinteilung auszunutzen, ist es daher zweckmaessig, die Struktur so zu verhaengen, dass die zerbrochenen Serieteile effektiv als Parallelen und die hintereinandergesetzten Parallelen wie eine Serie auftreten.
Zusaetzliche Boole´sche Variablen in den Records vom Typ HEADTYPE bzw LINKTYPE geben dann an, ob echte Parallelen bzw Serien vorliegen, oder Pseudeparallelen bzw Pseudoserien.

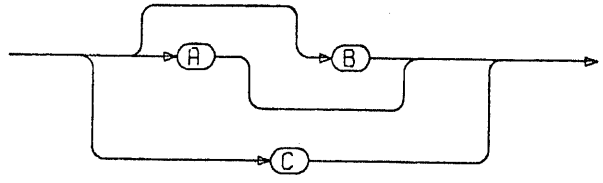
Beispiel 1:



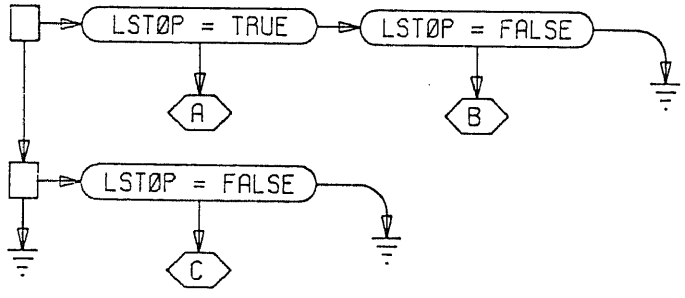
wird dargestellt durch:



Beispiel 2:



wird dargestellt durch (ohne Leersymbole):



Fuer das Brechen einer Serie sind also folgende drei Arbeitsgaenge zu durchlaufen:

1) Der Benuetzer hat verlangt, dass beim Punkt mit der internen Nummer x die Serie gebrochen werden soll:
Das Programm muss nun in der Serie-Parallel-Struktur den Zeiger auf dasjenige Serieelement finden, zu welchem die Punktnummer x gehoert.

Im referierten Record wird dann eine Boole´sche Variable geeignet gesetzt, welche den gewuenschten Bruch markiert.

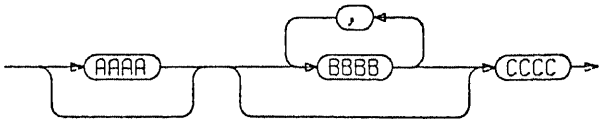
2) Das Programm durchsucht die Serie-Parallel-Struktur nach Bruchmarken.
Wenn solche gefunden werden, wird die Serie-Parallel-Struktur entsprechend umgehaengt und Boole´sche Variablen werden so gesetzt, dass die neue Serie-Parallel-Struktur richtig interpretiert werden kann.

3) In der Prozedur ASSIGNPOINTS wurde zu der Serie-Parallel-Struktur eine offene Struktur (OPENSTRUCT) aufgebaut, welche die Verbindungen der Punkte in vertikaler und horizontaler Richtung beinhaltet. Dabei wurde fuer die Verknuepfung keine Ruecksicht auf die Boole´schen Interpretationsmarken genommen:
Das Programm muss nun eventuell neue Punkte einfuehren, diese gemaess der Interpretationsinformation in der offenen Struktur einfuegen und gewisse alte Verbindungen modifizieren.

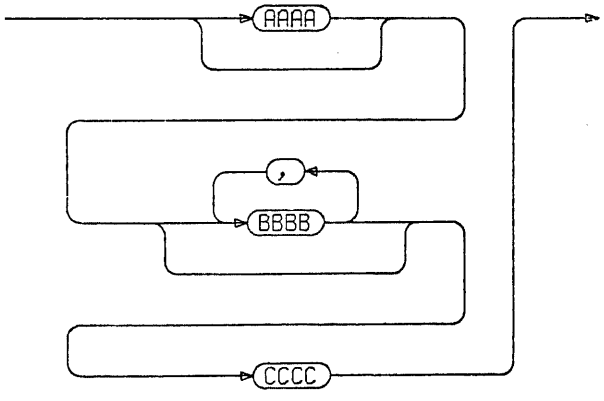
Analoge drei Arbeitsgaenge verlangt das Hintereinandersetzten von Parallelen.

Diese drei Aufgaben koennen durch je eine rekursive Prozedur geloest werden. Sie entsprechen im schematischen Aufbau den bereits bekannten Prozeduren und sollen an dieser Stelle nicht naeher betrachtet werden.

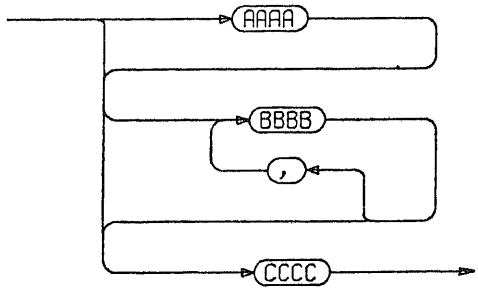
Die vier eben erwaehnten Operationen auf der Serie-Parallel-Struktur sind wohl die elementarsten und wichtigsten. Sie genuegen aber sicher nicht, um alle Benuetzerwuensche zu befriedigen. So waere es z.B nicht selten angenehm, wenn an Stelle des Diagrammteils:



oder:



nachstehender Diagrammteil erzeugt werden koennte:



Selbstverstaendlich ist es moeglich, auch diese Form noch zu markieren und zu interpretieren. (Worauf der Benuetzer seinen naechsten Wunsch anmelden wuerde).
Es ist aber zu beachten, dass der Implementierungsaufwand mit jeder neuen Form stark zunimmt, da nicht nur weitere Prozeduren benoetigt werden, sondern auch die schon vorhandenen fuer die bereits implementierten Formen komplizierter werden. Diese Interaktion der verschiedenen Formeninterpretationen werden im naechsten Abschnitt besprochen.

2.5 Interaktion der verschiedenen Interpretationen

In den Abschnitten 2.4 und 2.5 haben wir die Prozeduren kennengelernt, welche die primaere Interpretation der unmodifizierten Serie-Parallel-Struktur bewerkstelligen. Es sind dies im Wesentlichen die folgenden vier:

MINIMALLENGTH
: berechnet minimale Laenge der Unterdiagramme.

DISTRIBUTE
: verteilt den verfuegbaren Platz gleichmaessig.

LEVELS
: berechnet die Hoehenkoordinaten.

ASSIGNPOINTS
: generiert Punktnummern, berechnet Breitenkoordinaten, zieht Verbindungen.

Fuer die im Abschnitt 1.4 verlangten Modifikationen muessen zur Interpretation zusaetzliche Prozeduren eingefuehrt und die schon vorhandenen erweitert werden. Fuehrt man fuer eine erste Modifikation die zusaetzlichen Prozeduren ein, so werden auch diese wegen der weiteren Modifikationen auszubauen sein. Auf diese Art nimmt der Implementierungsaufwand mit jeder Modifikation staerker zu, und die einzelnen Prozeduren werden immer unuebersichtlicher. Gleichzeitig sind vom Programmierer immer mehr Faelle des Zusammenwirkens von Modifikationen zu beruecksichtigen.
Welche neuen Prozeduren durch unsere vier Modifikationsoperationen

bedingt und welche ihretwegen erweitert werden muessen laesst sich der folgenden tabellarischen Uebersicht entnehmen. Die Funktion jeder Prozedur wird unter jener Operation erklaert, welche ihr Vorkommen verschuldet.

MAIN

neue Prozeduren:
RELMINMAX
berechnet Hoehe und Tiefe eines Unterdiagramms bezueglich seines Hauptniveaus.

beeinflusste Prozeduren:
UPDOWNLINKS
CHANGEPARALLEL
CHANGESERIE
ASSIGNPOINTS
SERIEINTOPARALLEL
PARALLELINTOSERIE

Bemerkung: Die Operation MAIN ist fuer sich allein einfach zu implementieren (nur eine zusaetzliche Prozedur). Sie kompliziert aber viele der durch die anderen Modifikationen bedingten Prozeduren, und zwar die ohnehin schon uebersichtlichsten.

EXCHANGE

Diese Operation braucht keine neuen Prozeduren und beeinflusst auch keine.
Hier koennte der Eindruck entstehen, mit dieser Operation liessen sich nur Diagramme erzeugen, welche durch die Backus-Naur-Form direkt erhalten werden koennten. Dies ist aber nicht richtig, da sich auch mit eckigen Klammern [,] verlangte Leerkanten oder etwa Unterdiagramme D1 und D2 aus { D1 S D2 } vertauschen lassen.

BREAK

neue Prozeduren:
SIMPLIFYDIRPAR
normalisiert bei einelementiger, nichtatomischer Serie. D.h. vereinfacht die Struktur nach der zweiten Normalformenregel.
CHANGESERIE
haengt Teilserien in der Serie-Parallel-Struktur gemaess den BREAK-Marken zu Parallelen zusammen. Setzt die entsprechenden Interpretationsmarken.
OUTIFBROKEN
fuegt Leersymbole fuer Eingangs- und Ausgangspunkt zur Serie-Parallel-Struktur, falls die Serie auf der obersten Rekursionsstufe gebrochen wurde. (Fuenfte Regel fuer die Leersymbole).
SERIEINTOPARALLEL
interpretiert die gebrochene Struktur. D.h. fuegt neue Diagrammeckpunkte ein, zieht neue Verbindungen und veraendert alte.
Weitere Prozeduren (zur Behandlung von Leersymbolen) werden durch BREAK und APPEND gemeinsam bedingt und sind unten aufgefuehrt.

beeinflusste Prozeduren:
MINIMALLENGTH
RELMINMAX
DISTRIBUTE
ASSIGNPOINTS

APPEND

neue Prozeduren:
SIMPLIFYONEPAR
normalisiert bei Unterdiagrammen mit nur einer Serie. D.h. vereinfacht nach der ersten Normalformenregel.
CHANGEPARALLEL
haengt parallele Serien in der Serie-Parallel-Struktur gemaess den APPEND-Marken zu einer Serie zusammen. Setzt die entsprechenden Interpretationsmarken.
PARALLELINTOSERIE
interpretiert die hintereinandergehaengten Serien. D.h. fuegt neue Diagrammeckpunkte ein, zieht neue Verbindungen und veraendert alte.
Weitere Prozeduren (zur Behandlung von Leersymbolen) werden durch APPEND und BREAK gemeinsam bedingt und sind unten aufgefuehrt.

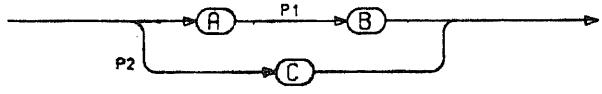
beeinflusste Prozeduren:
MINIMALLENGTH
RELMINMAX
ASSIGNPOINTS

Durch BREAK und APPEND gemeinsam bedingt:

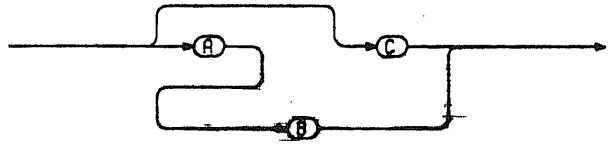
DELEMPTY
entfernt ueberfluessige Leersymbole.
UPDOWNLINKS
fuegt fehlende Leersymbole ein.

Da die einzelnen Modifikationsoperationen in vielfaeltiger Art zusammenspielen, habe ich mich bei der Implementation auf die mir am wichtigsten scheinenden Faelle beschraenkt, sodass fuer die Anwendbarkeit der Modifikationen folgende Ausnahmen gelten:

in der Situation

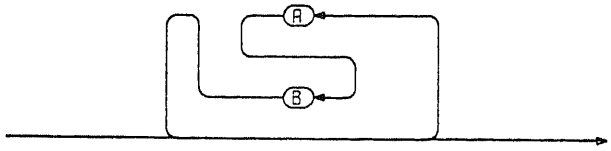


darf die Modifikationsfolge BREAK(P1); APPEND(P2) nicht auftreten. D.h. das Unterdiagramm



kann nicht generiert werden. (Das Programm unterdrueckt in diesem Fall den APPEND-Befehl).

Ebenso ist es untersagt, in rueckwaertsgefuehrten Unterdiagrammen BREAK oder APPEND anzuwenden. Also kann auch etwa das Diagramm



nicht erzeugt werden.

Wenn diese Einschraenkungen in der Praxis auch nicht besonders stoerend auffallen, so belegen sie doch einen Mangel an Systematik bei der Implementierung. Darueberhinaus legen sie den Verdacht nahe, dass der Implementierer eventuell einige Faelle des Zusammenwirkens von Darstellungsmoeglichkeiten uebersehen haben koennte.
Insbesondere wird es immer schwieriger, fuer die Korrektheit des Programms einzustehen, wenn noch mehr Interpretationsoperationen eingebaut werden sollten.

Die hier vermisste Systematik versuchen wir im Kapitel drei auf anderem Wege zu erreichen.

3. Systematische Erzeugung von Zeichenprogrammen

3.1 Synthese von Durchlauf- und Backus-Naur-Form Beschreibung

Der Vorteil der Durchlaufbeschreibung besteht in ihrer Moeglichkeit, verschiedenste Diagrammstrukturen zu formen.
Betrachtet man die Diagramme fuer eine bestimmte Grammatik, so sieht man, dass diese Moeglichkeit bei weitem nicht ausgeschoepft wird.
Die meisten Diagrammunterstrukturen sind Serie-Parallel-Graphen und die uebrigen Formen koennen durch eine kleine Anzahl von Durchlaufregeln beschrieben werden. Koennte man diese Formen einmal beschreiben und in der Eingabe fuer das Plotprogramm jeweils referenzieren, so waere der Nachteil des grossen Schreibaufwandes bei der Durchlaufbeschreibung eliminiert. Andererseits zahlt man diese Bequemlichkeit wieder mit dem Verlust an Flexibilitaet. Alle Diagramme muessen dann in den einmal eingefuehrten Formen ausgedrueckt werden.
Um auch noch die Vorteile der Serie-Parallel-Struktur bei der Breiteneinteilung zu nutzen, muss man noch Einschraenkungen bei der Definition der einzelnen Formen akzeptieren.
In der Eingabe der Plotprogramme der Endbenuetzer koennen die einzelnen Formen durch oeffnende und schliessende Klammern (eigentlich wuerden oeffnende Klammern genuegen) und Trennsymbole zwischen den Beschreibungen der Unterdiagramme verlangt werden, wie dies in der Backus-Naur-Form Eingabe z.B. durch
{ ´´1. Unterdiagramm´´ $ ´´2. Unterdiagramm´´ } geschieht.
Diese Klammern und Trennsymbole (Serielle Notation) koennen zusammen mit der Durchlaufbeschreibung und den formalen Unterdiagrammen in die Formenbeschreibung aufgenommen werden.

Obiges Formenbeispiel koennte also ungefaehr durch folgende Angaben umschrieben werden:

```
Delimiter fuer Metasymbole = ´.´ ;
 .{. ;
POSIT(IN); RIGHT; ´´1. Unterdiagramm´´; OUT; .$. ;
UP; LEFT; ´´2. Unterdiagramm´´; DOWN; IN
 .}.
```

IN und OUT bezeichnen im vorigen Marken fuer den Eingangs- bzw. Ausgangspunkt des Teildiagramms.

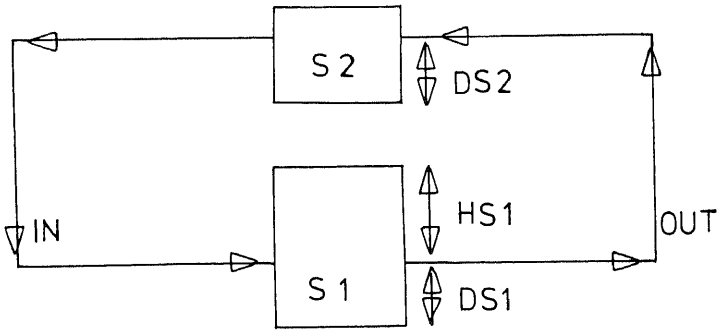
Die Breiteneinteilung des Diagramms wollen wir dem Programm ueberlassen; die Hoeheneinteilung jedoch durch Laengenangaben bei den Bewegungen UP und DOWN festlegen. Zu diesem Zweck muessen wir wissen, wie weit die Unterdiagramme nach oben und unten ueber das Verbindungsniveau hinausragen. Diese beiden Groessen sollen zu einem Unterdiagramm mit dem Namen S<i> automatisch liiert sein und die Namen HS<i> bzw DS<i> tragen. Hierbei ist <i> eine ganze Zahl aus einem noch festzulegenden Zahlenbereich.
Das Einfuegen eines Unterdiagramms S<i> in der Durchlaufbeschreibung soll durch die Nennung des Namens S<i> geschehen. Vorgaengig der ersten Nennung von S<i>, HS<i> oder DS<i> soll die Verarbeitung des Unterdiagramms durch die Anweisung NEW(S<i>) [oder REVERSE(S<i>)] bewirkt werden, da erst nach Behandlung des Unterdiagramms eine Referenz zur zugehoerigen Information sowie Hoehe und Tiefe bekannt sind. (Die

Anweisung REVERSE statt NEW bewirkt das Vertauschen der Bewegungen RIGHT mit LEFT und der Marken IN mit OUT.)
Die Metasymbole (Klammern und Trennsymbole) wie auch die Anweisungen NEW und REVERSE bestimmen also die Steuerung der Kompilation der Eingabe des Endbenuetzers, waehrend die Bewegungsanweisungen angeben, wie die Diagrammteile durch das Plotprogramm verbunden werden muessen.

Das oben beschriebene Beispiel kann definitiv angegeben werden durch:

```
STRUCTURE  DEL = .   ;
DECLARE S[1,2];
   .{.;
   NEW(S1); RIGHT; S1; OUT; .$. ;
   REVERSE(S2); UP(HS1 + DS2 + 1);
   LEFT; S2; DOWN; IN
ENDSTRUCT .}.
```

daraus ergibt sich das formale Diagramm:



[POSIT(IN) wird implizit generiert].

Im Wesentlichen entsteht dadurch eine Prozedur fuer die Behandlung einer Form. Fuer unser Beispiel ergibt sich folgender Rechenablauf:

0. Die Deklarationen seien verarbeitet und der Scanner hat das Metasymbol { bereits gefunden.
1. Generiere die Punkte mit den festen Marken IN und OUT, betrachte IN als aktuellen Punkt.
2. Rufe den Scanner.
3. Behandle die erste Unterstruktur (mittels Prozeduraufruf), setze den Zeiger S1.
4. Betrachte IN als alten , OUT als aktuellen Punkt.
5. Ziehe Verbindung von links nach rechts und haenge den Zeiger S1 an den Punkt OUT.
6. Teste, ob das Metasymbol gleich $ ist. Rufe den Scanner.
7. Behandle die zweite Unterstruktur (Prozeduraufruf).
8. Berechne die Hoehenkoordinate H.
9. Betrachte den aktuellen Punkt als alten Punkt, generiere einen neuen Punkt, betrachte diesen Punkt als aktuellen.
10. Ziehe Verbindung von unten nach oben. Setze die relative Koordinate H ein.
11. Wie Punkt 9.
12. Ziehe Verbindung von rechts nach links und haenge den Zeiger S2 beim rechten Punkt an.
13. Betrachte den aktuellen Punkt als alten Punkt und IN als aktuellen.
14. Verbinde von oben nach unten.

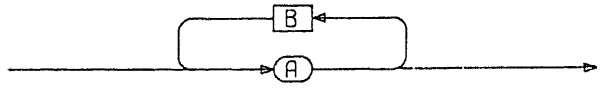
15. Teste, ob das Symbol gleich ENDSTRUCT ist. Rufe den Scanner.
16. Rufe die Prozeduren zur Bestimmung von HS<aktuelle Struktur> und DS<aktuelle Struktur>.

Diese Anweisungssequenz laesst sich leicht durch eine Makroentwicklung aus der praezisen Formbeschreibung erhalten. Der Programmierer ist fuer diese Aufgabe entbehrlich und kann durch ein entsprechendes Programm ersetzt werden.

Die nicht weiter zerlegbaren Strukturelemente seien hier vorlaeufig wieder Kaestchen mit Terminal- bzw. Nichtterminalsymbolen.
Mit solchen Symbolen und obiger Form koennen die nachfolgenden zwei Beispiele aufgebaut werden:

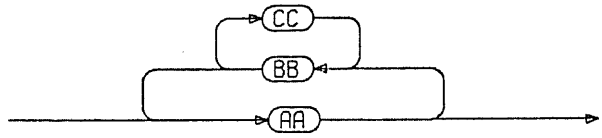
Beispiel 1

Die Eingabe { a $ <b> } fuer das Plotprogramm ergibt das Diagramm:



Beispiel 2

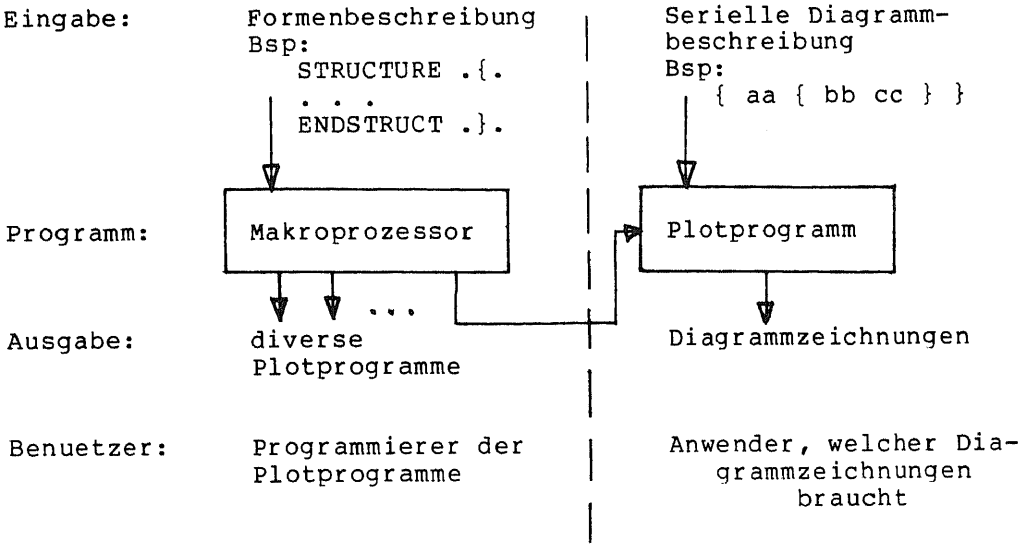
Die Eingabe { aa $ { bb $ cc } } fuer das Plotprogramm ergibt das Diagramm:



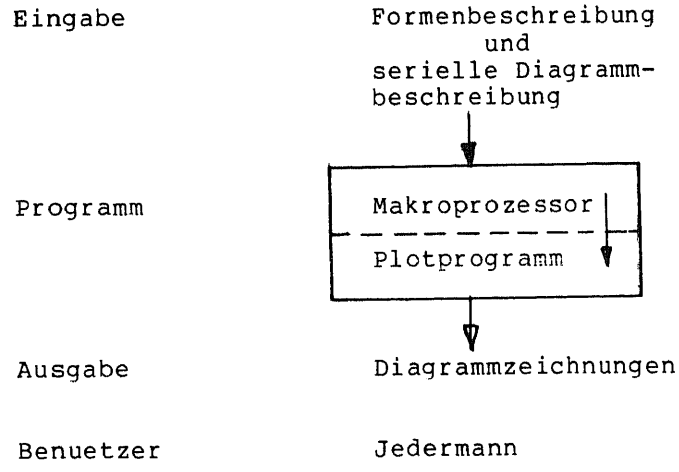
Bemerkungen:
---

1. Die Vereinbarung DECLARE ist noetig, damit die Formenbeschreibung in einem Pass uebersetzt werden kann.
2. Mit dem Schluesselwort DEL kann ein Delimiter fuer die Klammern und Trennsymbole definiert werden (Default = $).
3. Das Trennsymbol $ im obigen Beispiel ist ueberfluessig. Laesst man in der Formenbeschreibung .$. weg, so lauten die Eingabetexte fuer die Beispiele: { a <b> } bzw. { a { bb cc } } .

Die involvierten Programme samt ihrer Ein- und Ausgabe koennen dem naechsten Schema entnommen werden:



Die Trennung zwischen derjenigen Person, welche die Beschreibung der Formen in ein Programm eingibt und dem Endbenuetzer, der sich ein Werkzeug zur Diagrammerzeugung wuenscht, braucht nicht so rigoros durchgefuehrt zu werden, wie es dieses Schema andeutet. Es waere naemlich moeglich, jedem Benuetzer die Angabe der Formen selbst zu gestatten und somit ein Programm nach folgendem Schema zu handhaben:



Dieses zweite Vorgehen erlaubt dem Benuetzer, die Formen fuer seine Diagramme laufend seinen Beduerfnissen anzupassen.
Zudem braucht das Makroprogramm in diesem Fall kein unabhaengiges Programm zu generieren, da die Zeichnungen aus der seriellen Notation mit Hilfe der Formenbeschreibungen interpretiert werden koennten.
Die Liste der Nachteile des zweiten gegenueber dem ersten Schema ist etwas laenger und bewog mich zur Wahl der ersten Organisation:

1. Besonders rechenaufwendig waere ein Interpreter anstelle des Makroprozessors.
2. Das Plotprogramm wird haeufig neu generiert.

3. Das Plotprogramm wird haeufig kompiliert.
4. Das Programm fuer den Endbenuetzer ist wesentlich groesser, als fuer seine Anwendung notwendig.
5. Der Endbenuetzer muss eine zusaetzliche Anwendungssprache lernen, welche noch dazu komplizierter ist als die serielle Notation.
6. Der Endbenuetzer muss alle semantischen und implementationsbedingten Einschraenkungen der Formensprache kennen.
7. Dem Endbenuetzer bieten sich weitreichende Fehlermoeglichkeiten bei der Entwicklung der Formeneingabe.

Wie wir spaeter noch sehen werden, ist es vor allem der 6. Punkt, welcher es praktisch unmoeglich macht, das Makroprogramm einer breiteren Benuetzerschicht in die Hand zu geben.
Auf jeden Fall ist es aber moeglich, das vorhandene Programm MACRO unter Verwendung geeigneter Steuerbefehle an das Betriebssystem der Rechenanlage im Sinne des zweiten Schemas zu verwenden. Auf diese Art wurde das Programm waehrend der Entwicklung denn tatsaechlich betrieben.

Die Plotprogramme fuer den Endbenuetzer werden nicht vollstaendig aus den Angaben fuer die Formen generiert. Vielmehr fuegt das Programm MACRO zu einem festen Rahmenprogramm wesentliche Teile hinzu. Diese Teile bestehen vor allem aus einer Prozedur fuer jede Form, je einem Aufruf einer solchen Prozedur und einer Tabelle fuer die Metasymbole der seriellen Notation.
Die einzelnen Komponenten des festen Programmteils fuer alle Plotprogramme werden in einem spaeteren Abschnitt erlaeutert.
Die Elemente der Sprache zur Formenbeschreibung werden im naechsten Abschnitt vollstaendiger eingefuehrt.

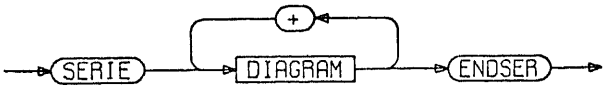
3.2 Die Sprache der Formenbeschreibung

Bisher haben wir zur Beschreibung der Formen Bewegungen, Serienamen, Metasymbole und die Marken IN und OUT verwendet.
Wie bei der Durchlaufbeschreibung 3 sollen bei Bedarf zusaetzliche Marken eingefuehrt werden koennen. Diese haben die Form M<nr>.
Um repetierte Teilformen guenstig auszudruecken ist eine Schleifenanweisung vonnoeten. Um nicht in jedem Schleifendurchgang einen neuen Markennamen verwenden zu muessen, sollen dieselben Markennamen mehrmals gebraucht werden. Dazu muessen einmal benutzte Namen wieder verfuegbar gemacht oder umbenannt werden.
Schleifen und Marken sollen mit den folgenden Beispielen eingefuehrt werden:

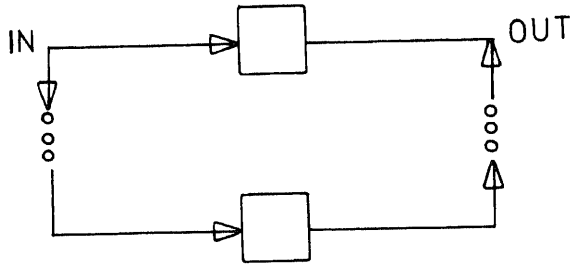
1) Es wird eine Serie dargestellt, welche mindestens ein Unterdiagramm enthaelt.

```
STRUCTURE
DECLARE S[1,1];
   $serie$;
   RIGHT; NEW(S1); S1;
   WHILE SYM = $+$;
     RIGHT; NEW(S1); S1
   END;
   OUT
ENDSTRUCT $endser$
```

Die Eingabe, welche vom erzeugten Plotprogramm als Serie gezeichnet wird, hat der nachstehenden Syntax zu genuegen:

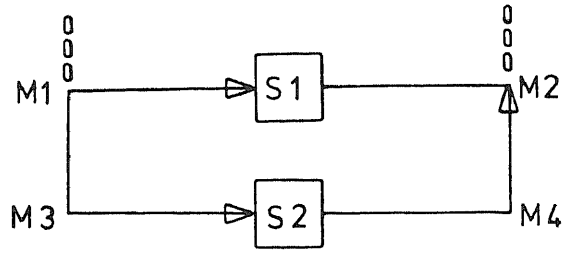


2) Es werden Parallelen dargestellt, welche mindestens eine Serie enthalten.



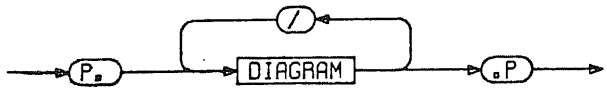

```
STRUCTURE
DECLARE S[1,2], M[1,4];
   $p.$;
   RIGHT; NEW(S1); S1; OUT;
   LNAME IN := M1; LNAME OUT := M2;
   WHILE SYM = $/$;
     NEW(S2); POSIT(M1); DOWN(DS1 + HS2 + 1); M3;
     RIGHT; S2; M4; UP; M2; SNAME S2 := S1;
     LNAME M3 := M1; LNAME M4 := M2
   END
ENDSTRUCT $.p$
```

Schematisch wird folgendes Diagramm konstruiert:

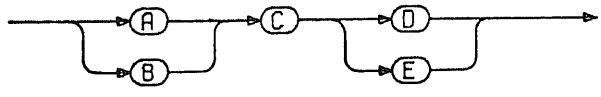


Am Ende der Schleife bekommt das Element der untersten Serie den Namen S1 und die Marken M3 bzw M4 werden in M1 bzw M2 umbenannt.

Die Eingabe zum Zeichnen von Paralellen genuegt der Syntax:



Wird beispielsweise ein Plotprogramm mit obigen beiden Formen generiert, so bewirkt die Eingabe
serie p. a / b .p + c + p. d / e .p endser
das Diagramm



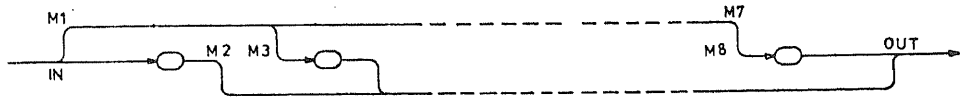
3) Im folgenden Diagrammschema haengt die Hoehe von IN nach M1 von den Hoehen aller Unterdiagramme ab. Da deren Anzahl nicht im voraus bekannt ist, werden diese in einer Schleife verarbeitet. Am Ende der Schleife muss das Maximum der Hoehen aller Unterdiagramme bekannt sein. Dieses Maximum kann in einer Variablen X1 fuer die Hoehe in der Schleife laufend berechnet werden durch:

```
X1 := 1;
WHILE  ... ;
  X1 := [X1, HS1]; ...
END;
 ...
M7; DOWN(X1); M8; ...
```

[Die eckigen Klammern bedeuten die Maximumfunktion].

Damit eine korrekte Breiteneinteilung bestimmt werden kann, muessen in diesem Beispiel M2 und M3 durch eine unsichtbare Kante verbunden werden. Diese verlangt man mit dem Parameterwert '-' bei einer Links- oder Rechtsbewegung:

```
... ; M2; RIGHT(-); M3; ...
```

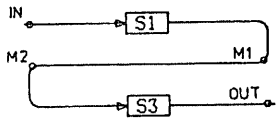


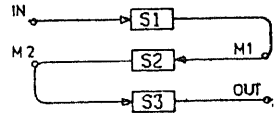
Wuerde die Kante von M7 nach M8 ebenfalls in der Schleife gezogen, so koennte die Hoehe von IN nach M1 formal eingesetzt und erst nachtraeglich berechnet werden:

```
 ... ; POSIT(IN); UP(VAR X1); ...
X1 := 1;
WHILE ... X1 := [X1, ..]; ...
END;
 ...
```

In diesem Fall wird derjenige Wert, der nach Verarbeitung der aktuellen Form in X1 steht, in die entsprechenden Koordinaten eingetragen.

Die zweite strukturierte Anweisung neben der WHILE-Schleife ist die IF-Anweisung. Diese kann dazu dienen, Varianten in der Darstellung einer Form nach Massgabe von Metasymbolen zu waehlen.
So koennen zum Beispiel die folgenden beiden Diagrammschemata





als Varianten derselben Form dargestellt werden durch

```
STRUCTURE
DECLARE S[1,3], M[1,3], X[1,2];
   $b.$;
   RIGHT; NEW(S1); S1; X1 := DS1 + 1; DOWN(VAR X1);
   M1; LEFT; X2 := 1;
   IF SYM = $-$;
     REVERSE(S2); X1 := X1 + HS2; S2; M2; X2 := DS2 + 1
   FI
   ELSE M2 END;
   UP(-); IN; POSIT(M2); NEW(S3);
   DOWN(X2 + HS3); RIGHT; S3; OUT; UP(-); M1
ENDSTRUCT $.b$
```

Die erste Variante wird gewaehlt durch den Input
b. ''1. Diagramm'' ''2. Diagramm'' .b
und die zweite durch
b. ''1. Diagramm'' - ''2. Diagramm'' ''3. Diagramm'' .b

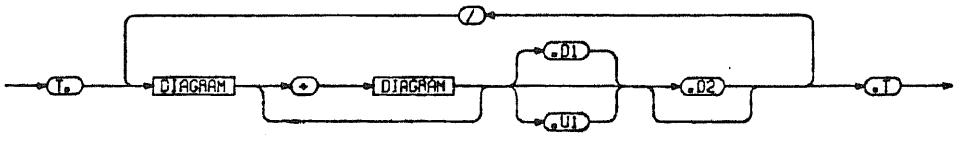
Da die Punkte mit den Marken IN und OUT nicht auf demselben Niveau zu liegen brauchen, muss festgelegt werden, welcher Punkt in der Form auf das Hauptniveau (Verbindungsniveau) gesetzt werden soll. Das kor-

rekte Verbindungsstueck zwischen zwei verbundenen Unterdiagrammen zu bestimmen ist dann die Aufgabe des Programms. Nach Konvention soll jeweils der Punkt mit der Marke IN das Hauptniveau festlegen. Wuenscht man aber den Punkt mit der Marke M<i> derart auszuzeichnen, so bedient man sich der Anweisung MAIN(M<i>).

An einem letzten, etwas komplexeren Beispiel wird der Gebrauch von Boole´schen Variablen illustriert:

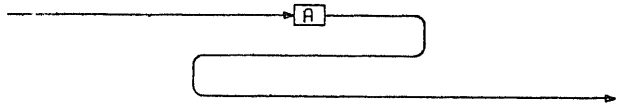
```
STRUCTURE
DECLARE M[1,4], S[1,2], B[1,3], X[1,1];
   $t.$;
   B1 := TRUE; B2 := FALSE; B3 := FALSE;
   LNAME IN := M1;
   WHILE B1;
     NEW(S1);
     IF B2;
       DOWN(X1 + HS1); M1;
       IF B3;
         POSIT(M3); DOWN; M2; POSIT(M1)
       FI;
       LNAME M3 := NIL; LNAME M4 := NIL
     FI;
     RIGHT; S1; M2;
     B2 := TRUE;
     IF SYM = $+$;
       REVERSE(S2); DOWN(HS2+DS1+1); M3;
       LEFT; S2; M4; X1 := 1 + DS2
     FI
     ELSE
       DOWN(DS1+1); M3; LEFT; M4; X1 := 1
     END;
     IF SYM = $.d1$;
       POSIT(M1); DOWN; M4
     FI
     ELSE
       IF SYM = $.u1$;
         UP; M1; POSIT(M4)
       FI
       ELSE
         UP(-); M1; POSIT(M4)
       END
     END;
     LNAME M1 := NIL; LNAME M2 := NIL;
     IF SYM = $.d2$; B3 := TRUE FI ELSE B3 := FALSE END;
     IF SYM = $/$; B1 := TRUE FI ELSE B1 := FALSE END
   END;
   DOWN(X1); RIGHT; OUT; UP(-, M3)
ENDSTRUCT $.t$
```

Die Syntax der Eingabe fuer das Plotprogramm ist gegeben durch:

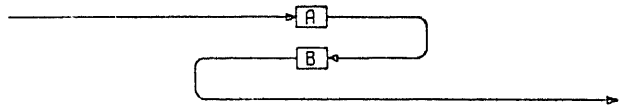


Die beschriebene Form ist anhand untenstehender Eingaben mit zugehoeriger Diagrammausgabe zu verstehen:

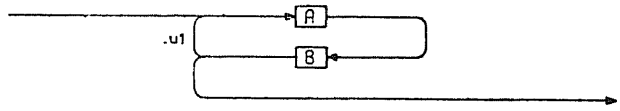
t. < a > .t



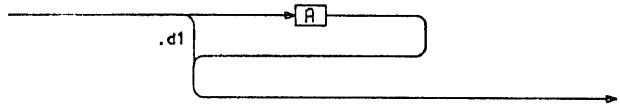
t. < a > + < b > .t



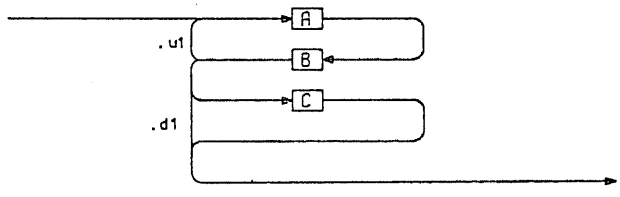
t. < a > + < b > .ul .t



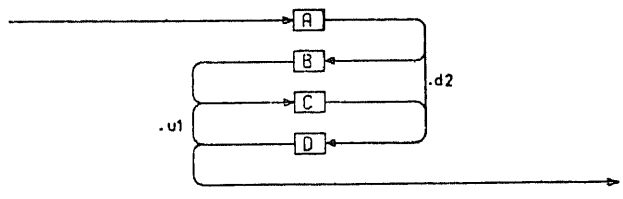
t. < a > .dl .t



t. < a > + < b > .ul / < c > .dl .t



t. < a > + < b > .d2 / < c > + < d > .ul .t



3.3 Der feste Rahmen der Plotprogramme

Der fixe Teil eines Plotprogramms, welcher vom Programm MACRO mit Prozeduren fuer die einzelnen Formen versehen wird, besteht aus

1) einem Scanner, welcher die Eingabesymbole prueft. Er arbeitet mit der Tabelle der Metasymbole, welche vom Programm MACRO zusammengestellt wurde.
2) einer Prozedur, welche eine Tabelle der Abkuerzungen der Wortsymbole fuer die Kaestchen erstellt.
3) Prozeduren zum Verarbeiten der atomischen Elemente der Struktur (bei Syntaxdiagrammen Terminal- und Nichtterminalsymbole).
4) Prozeduren, welche von allen Formenprozeduren aufgerufen werden, zum
   - Verbinden zweier Punkte,
   - globalen Speichern der Punkte der aktuellen Form,
   - Berechnen der relativen Niveaux bezueglich der Hauptniveaux.
5) einer Prozedur, welche jedes Diagramm initialisiert.
6) Prozeduren, welche die absoluten Hoehenkoordinaten aus den relativen Koordinaten berechnen.
7) Prozeduren, welche die Breitenkoordinaten bestimmen.
8) einer Prozedur, welche die Verbindungsstuecke zwischen aneinandergrenzenden Unterdiagrammen konstruiert.
9) Prozeduren fuer Plot und Prinplot der fertig beschriebenen Diagramme. Es sind dies im Wesentlichen dieselben Programmteile, die auch das Backus-Naur-Form Programm verwendet.
10) einer Prozedur, welche Steuerparameter fuer Art und Dimensionierung der Plotausgabe einliest.

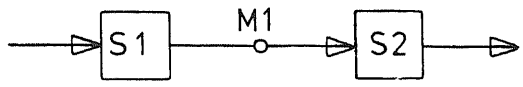
Unter diesen Komponenten gehoeren die meisten berechtigterweise zum unveraenderlichen Teil der Plotprogramme, da sie durch die Aufgabenstellung in dieser oder aehnlicher Art gefordert werden.
Die Partien 3), 7) und 8) hingegen schraenken die Menge der darstellbaren Figuren willkuerlich ein.

Zu 3)
Die atomischen Elemente brauchen sich nicht auf Terminal- und Nichtterminalsymbole zu beschraenken. Werden beliebige Figuren als Atome zugelassen, so kann man sich von der Einschraenkung auf Syntaxdiagramme loesen.

Zu 7)
Die Berechnung der Breitenkoordinaten bildet einen problematischen Teil des Programms. Vor allem macht schon der Begriff der ausgewogenen Einteilung Schwierigkeiten. Im Abschnitt 2.1 haben wir einen Algorithmus angetroffen, dessen Resultat den scheinbar vernuenftigsten und vermutlich einzigen fuer alle Graphen brauchbaren Einteilungsbegriff definiert. Dieser Algorithmus legt die Einteilung jedoch vollkommen fest und laesst dem aesthetischen Ermessen des Programmierers ueberhaupt keinen Spielraum.
Leider sind die Ergebnisse des Algorithmus fuer gewisse Graphenklassen wie z.B. Serie-Parallel-Graphen nicht sehr brauchbar.
Offenbar laesst sich der intuitive Begriff der ausgewogenen Einbettung fuer alle in Betracht kommenden Graphen nicht ohne Widerspruch konkretisieren.
Wegen der Bedeutung der Serie-Parallel-Struktur fuer die Syntaxdiagramme waere es wuenschbar, den oben erwaehnten allgemeinen Algorithmus mit einem solchen fuer serie-parallele Teilgraphen zu kom-

binieren.
Wegen der Inhomogenitaet und Komplexitaet des resultierenden Verfahrens habe ich diesen Vorschlag noch nicht implementiert.

Zu 8)
Bezeichnet beispielsweise in der Situation



S1 oder S2 ein nichtatomisches Unterdiagramm, so muss der Punkt M1 durch ein geeignetes Verbindungsstueck ersetzt werden.
Stellt man alle Faelle von moeglichen Verbindungen auf, so muss man beachten, dass der Ausgangspunkt von S1 nicht auf derselben Hoehe zu liegen braucht, wie der Eingangspunkt von S2. Ausserdem koennen beim Punkt M1 noch vertikale Kanten einmuenden.
Hier besteht die Moeglichkeit, den Programmierer seine Anschlussstuecke selbst formen zu lassen, oder diese Arbeit einem festen Programmteil zu ueberlassen. Ein entsprechender Teil fuer die Formensprache koennte ohne prinzipielle Schwierigkeiten implementiert werden. Dasselbe gilt fuer eine Verbindung von Unterdiagrammen in vertikaler Richtung, welche mit der gegenwaertigen Implementierung nicht ausgefuehrt werden kann.

3.4 Verallgemeinerung der atomischen Elemente

Ersetzt man die Kaestchen fuer Terminal- und Nichtterminalsymbole durch beliebige andere Figuren, so lassen sich Programme generieren, die Zeichnungen herstellen, welche im Wesentlichen an folgende zwei Formenprinzipien gebunden sind:
1. Die Verbindungskanten zwischen den atomischen Elementen verlaufen horizontal oder vertikal.
2. Die Zeichnungen sind rekursiv aufgebaut.

Bemerkung zu 1.

Im Gegensatz zu den Definitionen fuer Syntaxdiagramme brauchen die allgemeineren Zeichnungen nicht planar zu sein.
Auch muessen sie nicht zusammenhaengend sein (unsichtbare Verbindungen) oder genau zwei Eckpunkte vom Grad eins enthalten.

Bemerkung zu 2.

Durch die Rekursivitaet sind wir auf eine Klasse von recht einfachen Bildern beschraenkt, weil groessere Linienverflechtungen nicht moeglich sind, da ja von einem Unterdiagramm keine Linien zu anderen Unterdiagrammen fuehren. Es waere aber denkbar, solche Linien zwischen atomischen Elementen in verschiedenen Unterdiagrammen zu etablieren. Allerdings duerften solche Bildelemente bei der Koordinatenberechnung fuer die Gesamtfigur nicht beruecksichtigt werden.

Da die atomischen Elemente im Folgenden komplexe Figuren sein koen-

nen, wollen wir sie fortan mit ''Kaestchen'' bezeichnen.

Sind die Prozeduren fuer die Erzeugung der Kaestchen einmal geschrieben, so bietet es keine Schwierikeiten, diese in den uebrigen Programmkontext einzufuegen. Diese Prozeduren muessen dem Umgebungsprogramm einzig die Hoehe, die Tiefe und Breite des Kaestchens, sowie die Lage der Anschlusspunkte fuer die Verbindungslinien uebermitteln.
Nun waere es wohl sinnvoll, die Beschreibung der Kaestchen ebenfalls durch die Formensprache zu taetigen.
Diese Spracherweiterung muesste die elementaren Plotbefehle ausdruekken koennen.
Ausserdem sollte sie Schleifen und bedingte Anweisungen enthalten, um Repetitionen und Varianten zu erfassen.
Um den Programmierer durch eine solche Erweiterung nicht ueber Gebuehr einzuschraenken, wuerde man wohl zulassen, dass jene PASCAL-Text enthalten darf, welcher vom Programm MACRO in die Kaestchenprozeduren hineinzumischen ist.

Mindestens die elementaren Plotbefehle koennen sehr direkt in entsprechende Aufrufe von Plotroutinen uebertragen werden, so dass sich damit der Schreibaufwand fuer die Figurenbeschreibung kaum verringern liesse. Der Vorteil laege hier also darin, dass die Prozeduren automatisch an die richtige Stelle im Programm eingebaut und alle Anpassungen die im Programm noetig werden, ueberall fehlerfrei eingesetzt werden. Das Programm MACRO wird in diesem Fall insbesondere als Organisationshilfe bei der Programmierung benutzt.
Allzu einfach waere diese betrachtete Erweiterung aber doch nicht zu implementieren. Da der Layout der Figuren von Bedingungen und variablen Textlaengen abhaengt, sieht sich der Implementierer hier mit Problemen der Hoehen- und vor allem der Breiteneinteilung konfrontiert, wie sie bei der Durchlaufbeschreibung selbst auftreten.
Es draengt sich daher auf, die Darstellungsmoeglichkeiten mittels der Erweiterung der Formensprache bescheiden zu gestalten.

Mit verallgemeinerten Kaestchen kann MACRO zum Beispiel ein Plotprogramm erzeugen, welches dem Zeichnen von Diagrammen zur Illustration der Serie-Parallel-Struktur dient.
(Die vertikale Verbindung von Kaestchen muesste dann auch implementiert sein).
Die dazu erforderliche Durchlaufbeschreibung ist offensichtlich trivial.

3.5 Automatisch erzeugte Dokumentation der Plotprogramme

Jedem Anwender eines von MACRO erzeugten Plotprogramms muss mit dem Programm eine Gebrauchsanweisung uebergeben werden. Diese gibt Auskunft ueber die Form des Inputs und den dadurch determinierten Output. Das heisst, dem Anwender muss Syntax und Semantik der seriellen Notation beigebracht werden.
Tatsaechlich beschreibt man mit der Formensprache genau diese beiden Dinge. Gemaess unserer Unterscheidung von Anwender und Implementierer des Plotprogramms koennen wir die Formenbeschreibung aber nicht in die Gebrauchsanleitung aufnehmen. Zudem macht die Verquickung von syntaktischer und semantischer Information die Formenbeschreibung selbst fuer den Eingeweihten nicht besonders lesbar.
Es ist in diesem Kontext nicht an den Haaren herbeigezogen, die Syntax der seriellen Notation durch Diagramme zu bezeichnen.

Sind dem Programm MACRO diese Diagramme bekannt, so kann es Woerter generieren, welche durch diese Diagramme definiert sind. Diese Woerter ihrerseits werden vom (erzeugten) Plotprogramm zu Plotbeispielen transformiert, welche die Bedeutung der Eingabe bildlich umreissen.
Als erstes geht es demnach darum, aus der Formenbeschreibung die Syntaxangaben zu extrahieren und in Syntaxdiagramme bzw. die serielle Notation zur Beschreibung derselben umzuformen.

Die folgenden syntaxbestimmenden Anweisungen werden in die nebenstehenden Diagrammstuecke transformiert, falls man von der Verwendung von Boole´schen Variablen absieht:

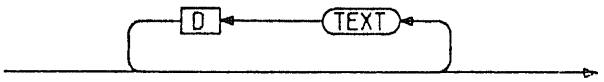
Ø) es gelte STATEMENT

1) $text$

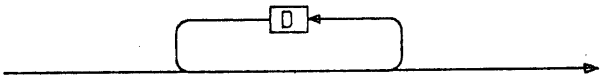
2) NEW(S<nr>)

3) REVERSE(S<nr>)

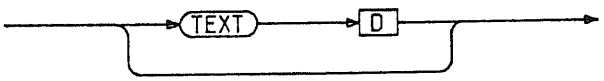
4) WHILE SYM = $text$;
   STATEMENT
   END



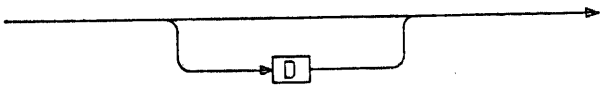
5) WHILE SYM <> $text$;
   STATEMENT
   END



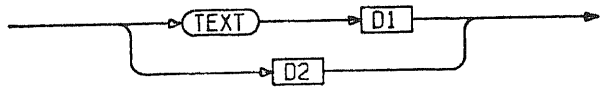
6) IF SYM = $text$;
   STATEMENT
   FI



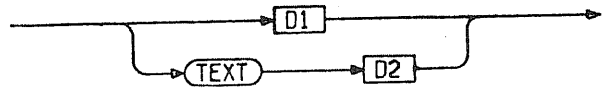
7) IF SYM <> $text$;
   STATEMENT
   FI



```
8) IF SYM = $text$;
      STATEMENT 1
   FI
   ELSE
      STATEMENT 2
   END
```




```
9) IF SYM <> $text$;
      STATEMENT 1
   FI
   ELSE
      STATEMENT 2
   END
```

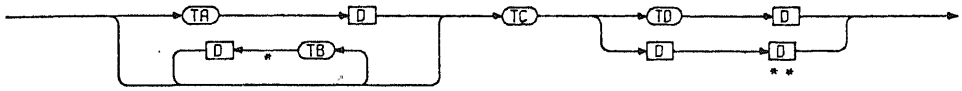


Diese Diagrammbestandteile bauen sich also aus Serien, Parallelen und Schleifen auf. Es ist demnach leicht, die vorigen neun Syntaxanweisungen in die serielle Notation eines Plotprogramms zu transformieren, mit welchem diese drei Formen dargestellt werden. Ein solches Plotprogramm koennte mit dem Programm MACRO generiert und in MACRO integriert werden.

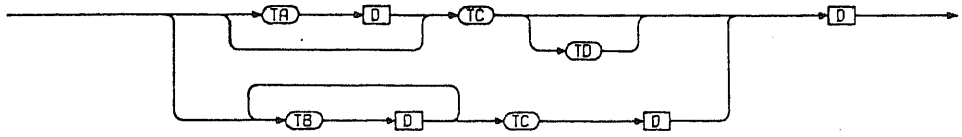
Undurchsichtiger wird die Sache, wenn WHILE-Schleifen und IF-Verzweigungen von Boole´schen Variablen abhaengen.
Diese Komplizierung soll an einem Beispiel erklaert werden:

```
STRUCTURE
   B1 := FALSE;
   IF SYM = $t1$;
     STATEMENT
   FI
   ELSE
     WHILE SYM = $t2$;
       STATEMENT;
       B1 := TRUE
     END
   END;
   $t3$;
   IF SYM = $t4$;
     STATEMENT
   FI
   ELSE
     STATEMENT;
     IF B1;
       STATEMENT
     FI
   END
ENDSTRUCT
```

Wuerden in diesem Fragment die eingerahmten Teile fehlen, so muesste die Eingabe fuer diese Form das folgende Diagramm erfuellen:



Die eingerahmten Teile bewirken nun aber, dass das Kaestchen ** genau dann zu durchlaufen ist, wenn * durchlaufen wurde.
Das korrekte Diagramm sieht deshalb so aus:



Man kann hier also statt des korrekten Diagramms, welches nicht ganz leicht aus der Formenbeschreibung konstruiert wird, ein anderes leicht bestimmen, bei dem die Terminalsymbole numeriert und die Kanten noch besonders markiert werden.
Jeder Terminalsymbolnummer entspricht dann eine Boole´ sche Variable (Durchlaufvariable) und jede Markierung stellt eine Disjunktion von Konjunktionen solcher Durchlaufvariablen oder ihrer Negierten dar.
Eine Kante mit Markierung L darf im Diagramm genau dann durchlaufen werden, wenn vorher eine Menge von Terminalsymbolen durchlaufen wurde, deren Durchlaufvariablen den Ausdruck L wahr machen.
Die Markierungsausdruecke werden gefunden, indem die Boole´ schen Variablen der Formenbeschreibung durch Disjunktionen von Konjunktionen von Durchlaufvariablen ausgedrueckt werden.
Mit diesen Ausdruecken und den Bedingungen der IF- und WHILE-Schleifen koennen die Markierungen zusammengesetzt werden.
Die markierten Diagramme sind natuerlich weit weniger praktisch als die unmarkierten. Sie haben aber den Vorteil, dass die mit einem festen Plotprogramm gezeichnet werden koennen.

Ausser der Syntax muss auch die Semantik der Form dem Benuetzer klargemacht werden.
Dies kann geschehen, indem man zur seriellen Notation fuer eine Form einige erzeugte Diagrammbeispiele zeigt.
Enthaelt die Formenbeschreibung weder Schleifen noch Verzweigungen, dann genuegt ein Diagramm zur Erklaerung der Form. Schleifen und Varianten vergroessern die erforderliche Anzahl der Beispiele fuer die Abdeckung einer Form betraechtlich.
Diese Beispiele koennen mit demjenigen Plotprogramm gezeichnet werden, welches illustriert werden soll. Dazu braucht nur die Eingabe in serieller Notation vom Dokumentationsprogramm erzeugt zu werden.

3.6 Offene Probleme

In den bisherigen Abschnitten war vor allem die getane Arbeit vorgestellt worden. Jetzt sollen einige Aufgaben aufgezaehlt werden, deren Erfuellung diese Arbeit abschliessen, oder Grundlagen fuer Erwei-

terungen dafuer schaffen. Die Abschlussarbeiten bestehen vorwiegend aus reinen Programmieraufgaben, welche das MACRO-Programm betreffen, und wegen ihrer Arbeitsintensitaet zurueckgestellt wurden.
Von diesen wuenschbar, bzw. unbedingt noetig ist:
- Ein Error-Recovery im Analyseteil fuer die Eingabe der Formensprache.
- Ein Error-Recovery, welches in jedem erzeugten Plotprogramm der Syntaxanalyse der seriellen Notation beigeordnet ist. Dieses Error-Recovery muss zum Teil im festen Stueck der Plotprogramme verankert sein, und zum andern Teil durch das Programm MACRO ins Plotprogramm eingefuegt werden.
- Die Implementierung des Breiteneinteilungsalgorithmus mit Freiplatzverteilung proportional zum Ausmass der Unterdiagramme entlang maximaler Bahnen.
- Die Moeglichkeit, Kaestchen auch vertikal zu verbinden.
- Eine semantische Analyse der beschriebenen Formen.
- Eine Erweiterung der Formensprache fuer Elemente etwa der folgenden Richtungen:
  i) allgemeinere Atome.
  ii) allgemeinere Anschlussstuecke.
  iii) Trennung von Verknuepfungslinien, welche die gegenseitige Lage der Unterdiagramme in einer Form bestimmen, von den Verbindungslinien, welche in der Zeichnung sichtbar sind.
  iv) verschiedene Qualitaeten von Verbindungslinien (ausgezogen, gestrichelt, doppelt, usw.).
  v) Wahl von verschiedenen Platzeinteilungsalgorithmen.
- Die Programmierung eines interaktiven Systems zum Zeichnen und Modifizieren der Diagramme auf einem Bildschirm, und Anpassung der Darstellungsprogramme an dieses System.

Neben diesen konkreten Aufgaben ist die Untersuchung einer Reihe von allgemeineren Problemen in unserem Zusammenhang von Interesse. Unter diesen sind zu nennen:
- Die Sprachendefinierungseigenschaften von markierten Graphen. Hier geht es darum, in Aequvalenzklassen bezueglich der definierten Teilsprachen Graphen als Repraesentanten zu finden, welche noch vorgegebene formale Eigenschaften besitzen, wie zum Beispiel Lookahead 1 oder kreuzungsfreie horizontal-vertikal-Einbettbarkeit.
- Einbettungsalgorithmen fuer Graphenersetzungssysteme. Gemeint sind Algorithmen, welche den Ersetzungsregeln zugeordnet sind. Die Schwierigkeit besteht dort in der Erfassung des globalen Effekts auf die Einbettung bei lokaler Ersetzung eines Teilgraphen.

4. Literatur

[1] U. Ammann: ´´Die Entwicklung eines PASCAL-Compilers nach der Methode des strukturierten Programmierens´´, Dissertation ETH 5456, Eidgenoessiche Technische Hochschule, Zuerich, 1975.

[2] K. Bucher: ´´Automatisches Zeichnen von Diagrammen´´, Benuetzeranleitung, Institut fuer Informatik, Eidgenoessische Technische Hochschule, Zuerich, Maerz 1977.

[3] K. Bucher: ´´Automatisches Zeichnen von Diagrammen´´, Dissertation ETH 5959, Eidgenoessische Technische Hochschule, Zuerich, 1977.

[4] H.-P. Frei: ´´THALES, an Interactive System and its Application to Teaching Programming´´, Dissertation, Universitaet Zuerich, 1975.

[5] E. Marmier: ´´Automatic Verification of PASCAL Programs´´, Dissertation, Eidgenoessische Technische Hochschule, Zuerich, 1975.

[6] H. Sandmayr: ´´Strukturen und Konzepte zur Multiprogrammierung und ihre Anwendung auf ein System von Datenstationen (Hexapus)´´, Dissertation, Eidgenoessische Technische Hochschule, Zuerich, 1975.

Berichte des Instituts für Informatik

*	Nr. 1	Niklaus Wirth:	The Programming Language Pascal
*	Nr. 2	Niklaus Wirth:	Program development by step-wise refinement
	Nr. 3	Peter Läuchli:	Reduktion elektrischer Netzwerke und Gauss'sche Elimination
	Nr. 4	Walter Gander, Andrea Mazzario:	Numerische Prozeduren I
*	Nr. 5	Niklaus Wirth:	The Programming Language Pascal (Revised Report)
*	Nr. 6	C.A.R. Hoare, Niklaus Wirth:	An Axiomatic Definition of the Language Pascal
	Nr. 7	Andrea Mazzario, Luciano Molinari:	Numerische Prozeduren II
	Nr. 8	E. Engeler, E. Wiedmer, E. Zachos:	Ein Einblick in die Theorie der Berechnungen
*	Nr. 9	Hans-Peter Frei:	Computer Aided Instruction: The Author Language and the System THALES
	Nr. 10	K.V. Nori, U. Ammann, K. Jensen, H.H. Nägeli, Ch. Jacobi:	The Pascal 'P' Compiler: Implementation Notes (Revised Edition)
	Nr. 11	G.I. Ugron, F.R. Lüthi:	Das Informations-System ELSBETH
	Nr. 12	Niklaus Wirth:	PASCAL-S: A Subset and its Implementation
*	Nr. 13	U. Ammann:	Code Generation in a PASCAL Compiler
	Nr. 14	Karl Lieberherr:	Toward Feasible Solutions of NP-Complete Problems
	Nr. 15	E. Engeler:	Structural Relations between Programs and Problems
	Nr. 16	W. Bucher:	A contribution to solving large linear systems
	Nr. 17	Niklaus Wirth:	Programming languages: what to demand and how to assess them and Professor Cleverbyte's visit to heaven
*	Nr. 18	Niklaus Wirth:	MODULA: A language for modular multiprogramming
*	Nr. 19	Niklaus Wirth:	The use of MODULA and Design and Implementation of MODULA
	Nr. 20	Edwin Wiedmer:	Exaktes Rechnen mit reellen Zahlen
	Nr. 21	J. Nievergelt, H.P. Frei, et al.:	XS-0, a Self-explanatory School Computer
	Nr. 22	Peter Läuchli:	Ein Problem der ganzzahligen Approximation
	Nr. 23	Karl Bucher:	Automatisches Zeichnen von Diagrammen

* out of print