16. SYNTAX-DIRECTED PRETTY-PRINTER

東京大学　工学部　　鶴田陽和
武市正人
和田英一

§1. はじめに

Algol のプログラムを書くプログラマは，begin と end の対応をつけ，またその間にはさまれている statement に対して indentation（段落付け のために行の開始位置を引っこめること）を行う等，読み易さのためにレイアウトを考えてソースプログラムを記述する。これを自動的に行なおうというのが，pretty-printer の着想の主眼点である。これまでにいくつかの方式が発表されているが，実際に implement されているところは少ないと思われる。しかし今後不可欠な computer-aided programming になると考えられる。

我々の方式は大きく2段に分けることができる（図1参照）。当初扱う言語は Pascal である。前段では，出力されたものがそのままコンパイラにかけられるという条件を守りながら，pretty-printer を使い，ソースプログラムに対して改行，indentation，spacing（空白の挿入）を行いプリティなレイアウトで出力する。後段では documention を主な目的として，multi-font のプリントルーチンを利用して下線を reserved word に引き，symbol をその計算機で implement されている文字に限らず全てマニュアル通りのものへと変換して出力する。これは，前段で出力された既に pretty-printing のすんだプログラムのリストをもとに，font 変換プログラム FONTCONV を用いて multi-font のプリントルーチンの入力となるようなリストを作ることによって行なわれる。これによってマニュアルどうりの完全なプログラムテキストの実現が可能となる。

§2. pretty-printer の動機とその意義

LISP や Algol 系の言語でプログラムを書くプログラマは，配置を考え

第17回プログラミング
シンポジウム　1976.1

ずにソースプログラムを書き並べることから発生する判読の困難さ——例えばソースプログラムの文字列を余分の空白をとらずに詰めていった場合の読みにくさを思い起こして頂けばよい——を避けるために，自分なりのプログラムのレイアウトの方法を実行している。

そのレイアウトの方法はプログラマにより様々であるが，普通採用されている基本的な手法として改行，indentation，spacingが考えられる。このみっつとも，ソースプログラムの文字列の順序に一切の変更を加えることなく，ただ冗長な空白を適宜削除，挿入することによって得られる。この様に同じ操作によっても得られるものではあるが，spacing

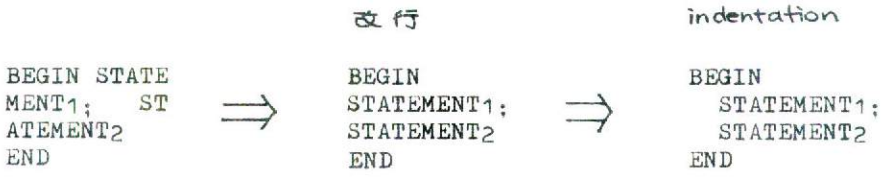


図 1

は delimiter や reserved word の前後への空白の挿入，indentation は段落付けのための行の前の方への空白の挿入と，ともに独立した意味が認められるので別なものとして扱える。

以上の様に今迄はプログラマが各自自分の流儀で，手で行っていたプログラムのレイアウトを自動的に行うのが pretty-printer であり，近年いくつかの pretty-printerが現われた。§3で，これ迄の代表的な prettyprinter と我々の製作したPascal用の pretty-printer，PASCAL-EDIT について若干の解説を行うが，大抵の場合はその手法に対して統一的な思想と呼べる程のものはなく，種々のプログラムの構造に対してとる hand-pretty-printing のレイアウトを想起して，そのレイアウトで自動的に編集されるように書かれている。それに対して我々が§4で提案する syntax-directed pretty-printerは，その編集の手法がひとつの原理に基づいており，いくつかの興味ある発想がその基盤となっている。

さて次に pretty-printing の意義にはどんなものがあるだろうか。その主なものとして，

1) プログラムの判読が容易になること

2) debug の助けとなること

3) documention に役立つこと

のみっつを我々は考えている.

ソースプログラムの読みにくさの解消が, pretty-printer の最初の動機であったが, それを追求してみると望ましいのはプログラムの意味や処理の流れのわかりやすいレイアウトである. プログラムの意味のわかりやすいレイアウトはそのアルゴリズムの理解の容易さから, アルゴリズム上の誤りの発見へと通じる. Goldstein は LISP の場合に於て, debug の補助手段としての意味を第一義に考えている[2]. また pretty-printer は構文をみていくことになり, その副次的な効果として, syntax-checker と重複する部分が存在することになり, 構文上の虫の発見と pretty-printer の密接な関係をみてとることができる.

その他に pretty-printer の主要な意義として, documention のための必要な技巧ということが挙げられる. このことに関して我々は, 本質的には空白の挿入だけの従来の pretty-printer の機能にとどまらず, 使用する計算機で使用可能な文字だけでなく, 必要な全ての symbol を用いてプログラミング言語のマニュアルにそもそも記述してあった形式でプログラムを出力することを考えている.

```
IF X>=1 THEN A(.I.):=X   ⇒   if x≥1 then a[i]:=x
```

図 2

§3. pretty-printer の例

この章では pretty-printer の代表的な例をいくつか紹介する.

(1) W.M.McKeeman の Algoledit

1965 年に提出された Algol 60 用の pretty-printer で, 我々の知る限りもっともふるいものである. その特徴をいくつか列挙してみると, (i) 一対の begin と end の内側の statement に対して indentation を行う. (ii) 宣言は新しい行に書かれる. (iii) コ

メントは，begin の直後以外は改行して新たな行から始められ，コメント継続行にはコメント文以外はでてこない．（iv）特定の basic symbolに対してはその前後にそれぞれに特有の個数の（大抵は 1～2 個の）空白が見易さのためにおかれる，等の pretty-printer の基本的なアイデアがいくつか実現されている．

（2） SOAP(Simplify Obscure Algol Programs)

これもやはり Algol 60 の pretty-printer であるが前者より多くの機能を備えている．我々の PASCALEDIT もこれと似ている部分が多い．プログラムの理解と検査をやりやすくすることをその目的としてあげているが，そのためのレイアウトの基本思想のひとつとしてプログラムの左端を眺めることによってプログラムの各部分の重要な特性を一目でわかるようにすることを主張している．例えば，代入文や宣言やラベルなどは行の先頭から始めるようにするのである．また条件文やブロックや手続きの占める範囲を表わすのに，indentation が使われている．これによってプログラムの階層的な構造や，入れ子になっている構造が直観的に把めるようになる．

SOAP には SOAP-C と言われる documentation のための拡張版があり改頁返抜える．例えば，ひとつの手続きがある頁の大部分を占めたときは，次の手続きはその頁の残りの部分をとばして新たな頁から開始される．（最新版は SOAP-PLUS といわれる．）

（3） Ira Goldstein の pretty-printing

ここでは pretty-printing という言葉が登場する．debug の補助手段として LISP のプログラムに適当な改行と indentation を行い読み易い format へ変換して出力するのがここでの目的である．まず，図3のような3種類の format を考える．

この3つの format を使い，行の中が有限であることを考慮に入れ，プリティでありしかも全体のたての長さができるだけ短いレイアウトで出力するアルゴリズムを考える．Goldstein はプログラムを tree に表わし，ある subtree を出力するのにはどの format

が適当かを見積っていく pretty-printerで，計算時間が普通の原始関数 PRINT の数倍ですむものを提案している．

linear format

(<FUNCTION> <ARG(1)> <ARG(2)> ... <ARG(N)>)

standatd format

(<FUNCTION> <PRETTY*PRINT ARG(1)>
<PRETTY-PRINT ARG(2)>

<PRETTY-PRINT ARG(N)>)

miser fofmat

(<FUNCTION>
<PRETTY-PRINT ARG(1)>

<PRETTY-PRINT ARG(n)>)

図3

(4) PASCALEDIT

これは今年の春に Pascal用の pretty-printer として書かれたもので現在東京大学工学部情報工学の FACOM 230-38 で実用に供している．この pretty-printerは，異なる basic symbolや様々な構文的な構造のひとつひとつに対してプリティなレイアウトで出力する手続きがひとつづつ対応していて，改行，indentation，spacing を行う．行の長さは有限であるため，indentation の使用には限度があるがある桁を越えると indentation は起こらないようにしてこの問題を解決している．図4にその主な仕様をあげておく．

LISP の pretty-printer としては semantical な文脈を考慮に入れずに括弧を眺めるだけの簡単なアルゴリズムによる レイアウト comfortable output format [8] もある．

他に，[5]，[7]がある．

§4. Syntax-Directed Pretty-Printer

前述の PASCALEDIT に於て懸案となっていた点をあげてみる．

(i) 例えば代入文のように我々が simple statementと呼んだものが2行以上にわたる場合の，行の切りかたはプリティではない．図5

(i) begin, end の 対応

```
begin statement1; statement2; ... ;statement-n  end
```

⇒

```
begin
   pp-ed statement1;
   pp-ed statement2;
            .
   pp-ed statement-n
end
```

(ii) if 文

```
if predicate then statement1 else statement2
```

⇒

```
if pp-ed predicate then pp-ed statement1
else
   pp-ed statement2
```

又は

```
if pp-ed predicate then
   pp-ed statement1 which cannot be written in one line
else
   pp-ed statement2
```

(iii) ラベル

```
label:something
```

⇒

```
label: pp-ed something
```

又は

```
label:
   pp-ed something which cannot be written in one line
```

(iv) 括弧

```
(***********...
*******)
```

⇒

```
(***********...
 *******)
```

図 4

のように数式は順に詰めることしか考えていない．このように我々が pretty-printing を考えなかった場合についてもプリティなレイアウトで出力したい．

```
(aaa+bb)/(c-(dd+ee)
*f-ggg)*(x+y)
```

図 5

(ii) documentation のことを考えて改良も pretty-printing の手段のひとつでしたい．例えば，ひとつの手続きはなるべく頁の始めからはじまり，まとまった頁におさまるようにしたい．

(iii) 特定の言語に把われずに pretty-printing の原理を追求してみたい．そして，もしPascal の pretty-printer が書かれたならば，それは他の言語の pretty-printer へ容易に書き換えられるものであってほしい．

以上のような問題点のひとつの解決策としてこころみたのが，syntax-directed な方法である．以下にその説明を行う．

今望ましいのはプログラムの意味や制御の流れを的確に表わしているレイアウトである．しかしこれらはプログラムの構文的な構造に大体あらわれているとみてよいのではないだろうか．確かにこれまでの pretty-printer もプログラムの各部分の syntax を解析して pretty-printing を行なっていることが多い．それならばプログラムの構文だけみていってもかなりのことができるのではないだろうか．そこでまずプログラムを <program> から terminal symbol に至る迄の構文の tree に表わしてみる．tree ができたとき各節点が semantical には何を表現しているかを一切考慮に入れない pretty-printer を考えてみる．

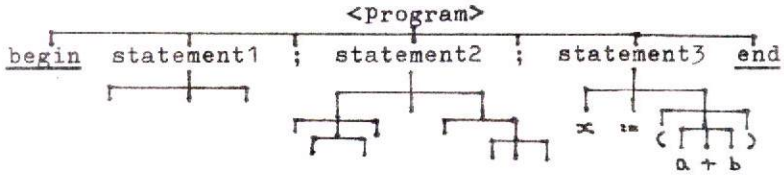


図6

(1) ある subtree が一行におさまるときはそのようにする．

(2) もし一行におさまらないときはその subtree を表わす節点のひとつ下のレベルを眺めていって下図のようにする．

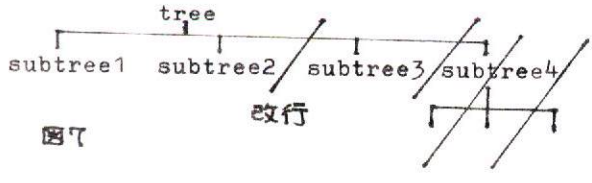


図7

```
subtree1 subtree2
  subtree3
  pretty-printed subtree4 which
    is indented on the second
    line
```

つまりひとつ下のレベルの節点を左から順にみていき，もしその節点で表現されている subtree を書き出すのに一行ですむときは書き出して

しまい，同じレベルにある次のsubtree も続けてその行におさまるならばそれも続けて書き出す．そうでないときは，2番目のsubtree は新たな行から書き出す．今ながめているレベルで一行に書き出せないsubtreeがでてきたら，改行して新しい行から（2）の手続きをrecursive に加えていく．

こうやって出力したプログラムがどこで改行を受けているかを考えてみると，構文上まとまっているところは離れたがらず，つながりのうすい箇所から改行を受けることになる．つまり構文上まとまっている部分はなるべく一行に，それも関連の強い部分から一行に書かれ，改行は構文的な要素の切れ目を表現することになる．先程の図5の場合は下図のように変わり目的とした改行ができたことになる．

図 8

```
(aaa+bb)
/(c-(dd+ee)*f-ggg)
*(x+y)
```

（3） 次に indentation について考えてみよう．例えばPASCALEDITではプログラムのsemantics に対して，つまり個別の構文的な構造に対しindentation のレイアウトを決定した．indentation についてよく考えてみるとその意義はプログラムの入れ子の構造を表わすことにて，ある意味的にまとまりのある部分の範囲を示すことである．ここで我々にひとつの啓示を与えてくれたのはKnuthの tree 構造 の導である．彼は

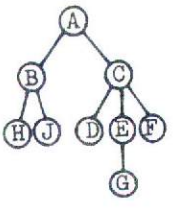


図 9

図9 のように indentation は tree 構造のひとつの表現であることを示している．そこで（2）のようなindentation のアルゴリズムを実行すれば，プログラムテキストに於ける indentation は構文の木に於けるレベルの表現となりある tree の範囲を示すものとなる．

（4） 上述の様にプログラムの tree を作り各節点を書き出すのに必要なスペースを計算した上，その tree をみて改行，indentation を行いながら書き出せば syntax-directed pretty-printing ができそうだが，

ここでひとつだけ問題が起こる．例えば括弧を<factor>と同じレベルの節点であるとみると図10のような少し不自然なレイアウトになってしまうのである．この問題はsemicolonや演算子の場合にも起こる．そこで

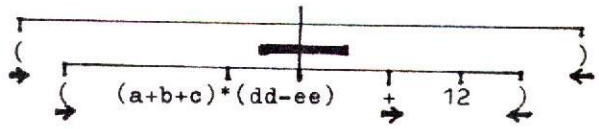
図 10

```
(
 (
 (a+b+c)*(dd-ee)
 +12)
)
```

```
(((a+b+c)*(dd-ee)
 +12))
```

図 11

図 12



構文の木にひとつだけ工夫をいれて左括弧にはそのレベルに関係なく書き出される順次のbasic symbolにくっつこうとする性質があり，右括弧についてはその逆であるとして出力すれば図11のようになりこの問題は解決される．つまりあるbasic symbolには出力に際して（２）の改行のアルゴリズムに優越して前か後にくっつきやすいという属性があるとみるのである．例えば演算子なら前，semicolonなら後である．

以上（１）～（４）のアルゴリズムによってsyntax-directedな方法が可能となったが，これは個々のプログラミング言語に依っていず数式の問題も解決し，又改行のアルゴリズムを単純に拡張することによって改良も可能となり前記の問題点にひとつの解決を与えている．なおここで論じなかった問題点に，BNF風の文法の記述とpretty-printer向きの文法の書換えとの関係，実際には<program>のtreeを全部作るわけにはいかないのでそれを切抜けるにはどうすればよいか等がある．

§５．Multi-Font印刷プログラム

前段pretty-printerで，改行と空白を適当に入れた出力はラインプリンタやカード用文字によるいわゆるハードウェア表現になっている．LISPのS式のようにハードウェア表現のプログラムはそれで処理がおわりだが，M式のように小文字，矢印を使うものや，Algolのreference言語のようにいろいろな字体やunderlineを使うものを，なるべくそれらしく出力したいときにはラインプリンタでは不充分である．CMUなどではXGPでゴシックにまるbegin，end等を出力したり，BTLでは植字

機械で出力したりしているが，我々はそのような高価なものは使わずタイプエレメントの交換できるミニコン制御の IBM タイプライターで印刷することにしている．現在 Pascalについてテスト中である．

これに使用するmulti-font 用タイパーのソフトウェアは実は3年程前に作ったもので，最初single-font で右側を揃えたり， indentation をかける程度であったもの［夏のミニコンのシンポジウム報告集参照］を直ちに機能増強し， multi-font版にしたものである［10］． multi-font にするためには，font-shift code をちりばめたテキストをテープから読みこみ，一頁分のイメージをミニコンのメモリーの中に作りあげ，テレタイプからどの font をセットせよという指令を出す．オペレータはそのエレメントをセットし，タイプライター用紙を出発の位置にあわせてミニコンに合図をするとその font の文字の部分が用紙に印刷される．次の font で印刷すべき部分があればまた指令が出，なければテキストの次の部分が読みこまれるというものである．この方式では例えば数式の添字は別の font と思い，用紙を少し上に合わせることでうまく印刷できる．

Pascal のプログラムの印刷ではどの種類の文字をどの font にするかをきめるのが大いに趣味によるところで， begin, end 等はゴシックにするという案もあるが，字体の太さが違和感がちなので一般の文字との区別がつきにくく，今はイタリックに underline となっている．名前の文字はイタリック，数字は APL の立体のものを使ってみている．これはテープ作成のプログラム FONTCONV の変更で簡単になおせる．演算子等は APL の場合が多く， colon や semicolonはパイカ，注釈用のカーリーブラケットだけはシンボルのエレメントにしかなくそれを用いた．この組み合せで図15 の過程をへて印刷したものを図 14 に示す．

さてテストをしてみたら思わぬ問題がひとつ生じた．それはもとのソフトウェアが右側を揃える方式のもので，今はその機能をオフにして使うが，このとき出力は入力の配置のままになっている．元の配置の方は font-shift codeが沢山入るために一行に本文があまり入らずに改行されている．するとタイプライターに出るプログラムも大変短くなってしまう．そこでタイプライター上での改行の指令コードを改たある必要があ

るのだが，いっそのこと pretty-printer 後段専用の multi-font 印刷プログラムを PLAN で書こうと計画中である．

図 13

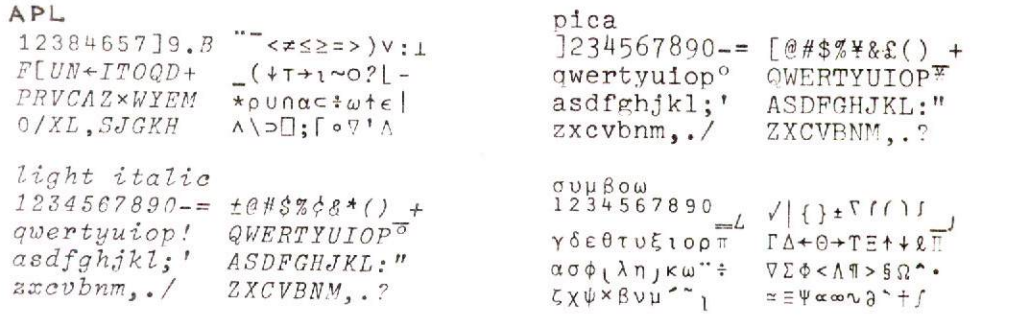


図 14

```
program spanningtree(output);
const n=6;
type node=1..n;
  setofnodes=set of node;
  graph=array[node] of setofnodes;
var g,a: graph;
procedure spantree(g: graph; s,b: setofnodes);
  var i,j: node;
  begin
    if s=[1..n] then
      begin
        for i:=1 to n do
          begin write(i: 5,'(');
            for j:=1 to n do
              if j in a[i] then write(j: 2);
            write(')')
          end;
        writeln
      end
    else
      for i:=1 to n do
        if i in b then
          begin b:=b-[i];
            a[i]:=g[i]*s;
            g[i]:=g[i]-s;
            spantree(g, s+[i], b+g[i]);
            if g[i]=[ ] then goto 1
          end;
    1:
  end;
begin
  g[1]:=[2];
  g[2]:=[1,3,4,6];
  g[3]:=[2,5,6];
  g[4]:=[2,5,6];
  g[5]:=[3,4,6];
  g[6]:=[2,3,4,5];
  a[1]:=[ ];
  spantree(g,[1],g[1])
end.
```

label 1;

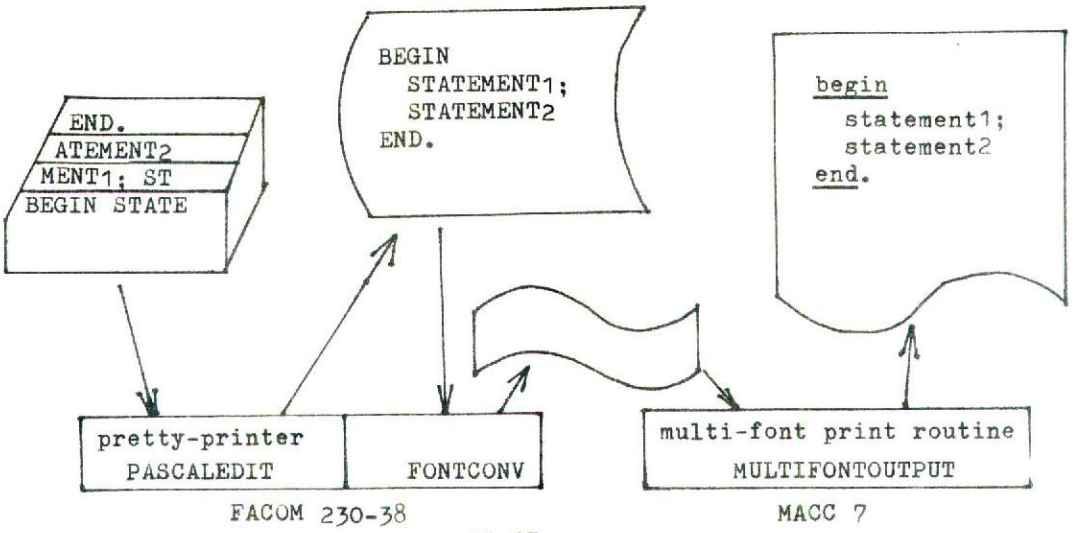


図 15

文献

1 Dahl,O.-J., Dijkstra,E.W. and Hoare,C.A.R., Structured Programming, Academic Press, 1972.

2 Goldstein,I., Pretty-printing, Converting List to Linear Structure, Artificial Intelligence Memo, no. 279, M.I.T., February 1973.

3 Knuth, D. E., The Art of Computer Programming, vol. 1, Addison-Wesley, 1968.

4 McKeeman, W. M., Algorithm 268, Algol60 Reference Language Editor, Communication of the ACM, vol. 8, no. 11, November,1965, pp667-668.

5 Mills, H. D., Syntax-Directed Documention for PL360, Communication of the ACM, vol. 13, no. 4, April 1970, pp216-222.

6 Scowen, R.D., Allin, D., Hillman, A. L. and Shimell, M., SOAP--A program which documents and edits Algol 60 programs, Computer J., vol. 14, no. 2, , pp133-135.

7 原合治他, ストラクチャードプログラミング支援システムSPOTとその評価, 第16回プログラミングシンポジウム報告集, 情報処理学会, 1975, pp 68 ~ 80.

8 Takeuchi, I.,and Okuno, H., A List Processor LIPQ, Second USA-Japan Computer Conference, 1975, Session 20-1-1.

9 鶴田陽和, プログラム印刷プログラム ― プリティ・プリンティング, 卒業論文, 東京大学工学部計数工学科, 1975.

10 Wada, E., Kakehi, K.,and Takeichi, M., An Analysis of Program Making, Programming Teaching Techniques, ed. W. M. Turski, North Holland Publ., 1973.