PRETTY-PRINT

by

Bob Boyer.

Memo No 64

1973

Department of Computational Logic
School of Artificial Intelligence
University of Edinburgh

A B S T R A C T.

A program for printing list structures in a pretty format is described. The program is fast, does only a little consing, and is exact (i.e., uses no heuristics).

PRETTY-PRINT.

Many programmers who deal with lists have found that a facility for pretty-printing list structures is a great convenience. (The only alternative to pretty-printing that I know of is a 'parenthesis count' which prints below each parenthesis the number of unbalanced parentheses to the left.) This note describes a pretty-print algorithm.

An apology is perhaps due before I describe the algorithm. The first pretty-print program I used was called GRINDEF; it was the standard at the MIT AI Laboratory. GRINDEF worked reasonably well on small formulas but was actually compute-bound on some of the larger formulas generated by theorem-proving programs I helped write. (I mean, the program would sometimes stop printing on our visual display for a few moments while it calculated.) That spurred me on to write an algorithm myself. I inspected the code for GRINDEF, and I can testify from that vision alone that anyone who thinks pretty-printing is trivial has just not thought about it long enough. The beauty of GRINDEF is that it does no consing. As a result, it was almost constantly re-computing what it might have remembered.

My first pretty-print did a fantastic amount of consing: three (gasp) times as much store as the formula being printed. But at least it worked faster, at least 10 times faster on large formulas. However, my program was still far from exact (I will explain precisely what I mean by 'exact' later). I now see that exactness was missing because I had not yet imagined the full complexity of the task. Let me try to give you an intuition of how complicated an exact pretty-print is: it is the only recursive function I know of with 9 really local variables.

Working in Edinburgh on another theorem-proving program, I found it necessary to write another pretty-print for list structures, this time in POP-2 instead of LISP. My current pretty-print does a small amount of consing, is very fast, and is completely exact.

a) The amount of consing done by the program is approximately one cons for every three lines of output.

b) The speed of the program is directly proportional to the size of the formula being printed; it takes about three times as long to pretty-print a formula as it does simply to print it. (This time is calculated for writing to a 'device' which uses up no time whatsoever.)

c) By exactness I mean that no 'heuristics' are used; the program knows precisely how much space every part of the formula needs for the several varieties of indentation before anything is printed.

Requirements of a Pretty-Printing Algorithm

Before beginning a description of what I want in a pretty-print program, let me make a few simplifying assumptions:

1) No special attention will be paid to things like LAMBDA's, PROG's or comments.

2) Nothing of the form ((...) ...) is permitted (i.e., we have always atoms in the function position).

3) Nothing of the form (FOO . BAR) is permitted.

All of these simplifications can be easily removed; the only reason I make them is to help keep a complicated discussion a little clearer.

The first requirement in a pretty-print algorithm is that it print the formula all on one line if it will fit.

The second requirement is that it print the formula with the arguments directly underneath one another if the formula will not all fit on one line. The best type of indentation is of the form:

```
(FOO (BAR1.........................)
     (BAR2........)
     (BAR3..............))
```

That is, with the first argument on the same line as the function symbol.

The other type of indentation is

```
(FOO
 (BAR1.....                    )
 (BAR2.....          )
 (BAR3.....          ))
```

The third requirement in a pretty-print algorithm is that if the second type of indentation must be employed, then the arguments will be moved as far to the right as possible. Thus

```
(FOO
   (BAR1.....                  )
   (BAR2.....          )
   (BAR3.....          ))
```

is better than the immediately preceding output because the arguments are indented further. Of course, the only reason that the second type of indentation must be employed is that if the first variety is used, then the output will go beyond the right-hand margin.

The pretty-print algorithm to be described in this note achieves these three requirements in the following way:

1) Any sub-formula that can be printed flat (on one line) is so printed flat, even if that entails the worst variety of indentation for the formulas which contain the sub-formula.

2) Subject to constraint (1) above, if a sub-formula cannot be printed flat it will be printed with its arguments indented in the best way or as far over as possible (if the second type of indentation is necessary). This indentation will be allowed for the sub-formula even if it entails the worst variety of indentation for the formulas which contain the sub-formula.

3) At the very worst, the arguments of each sub-formula will be indented at least one space further than the sub-formula.

(Unless, of course, the formula simply cannot be pretty-printed. I have never seen this happen with a line width of 80 characters.)

The implications of 1, 2 and 3 can be easily apprehended by inspecting some sample output. See the appendix.

The Pretty-Print Algorithm

The pretty-print algorithm is called PPR. It takes 3 arguments:

MARG1 which is the number of spaces you want in the left-hand margin,

the FMLA to be pretty printed,

and RPARCNT which is the number of characters you want to print on the last line of the pretty-printed text after the pretty-print is finished.

Normally RPARCNT is originally 0. The global variable MARG2 is set to the maximum number of characters that may be printed on a line. PPR expects that it can begin outputting immediately: i.e., the print head is positioned where pretty-printing is to begin. PPR does not print a new line after it is finished. You can still print as many as RPARCNT characters before exceeding MARG2.

Essentially, PPR works by making two complete passes through FMLA. The first pass is made by PPR1 which does no output whatsoever. Instead PPR1 simply calculates exactly where, what kind of, and how much indentation is required. The information returned by PPR1 is used by PPR2 which makes the second pass through FMLA. PPR2 does no calculating but does all the printing.

PPR1 returns its answers through four global variables: FLATSIZE, REMAINDER, STARTLIST, ENDLIST.

FLATSIZE may be a positive integer or false. If it is a positive integer then it is the number of characters it takes to print FMLA flat; furthermore FMLA can be printed on one line. If FLATSIZE is a positive integer, then STARTLIST and ENDLIST are of no significance. If FLATSIZE is false then FMLA cannot be printed on one line and information about where indentation should occur is found in STARTLIST and ENDLIST.

STARTLIST is a list of numbers. Each number is actually a packed triple representing three numbers. These numbers describe the kind and amount of indentation required for each sub-formula with indented arguments.

1) The bottom number is a kind of pointer to the sub-formula with indented arguments.

2) The middle number is the number of spaces further in that the arguments are to be indented.

3) The top number is a one bit flag indicating what kind of indentation is to be used. If the bit is on, then the first argument is to be printed on the same line as the function symbol; otherwise the first argument is to be printed on a new line.

The basic idea is that PPR1 returns an appropriate STARTLIST and that PPR2 will take this list and begin printing the FMLA the way that, say, PRINT does. But as PPR2 encounters each sub-formula, PPR2 asks whether the sub-formula is pointed to by the next entry on the STARTLIST. If so, the sub-formula is printed with its arguments indented according to the other information in the first entry on the STARTLIST. If the sub-formula is pointed to by the first entry on STARTLIST, then STARTLIST is CDR'd before the arguments are printed (recursively by PPR2 of course). If the sub-formula is not pointed to by the first entry on the STARTLIST, it is printed flat.

What kind of pointers are these 'kind of' pointers? They are the integers associated with each sub-formula by the depth-first enumeration of the sub-formulas of FMLA. Every time we enter PPR1 recursively we increment a global variable GRECCNT by one. The value of GRECCNT on entry to PPR1 is thought of as pointing to the current sub-formula that PPR1 is working on. Of course, when we reinitialize GRECCNT to zero and begin PPR2 (where we also keep uping the count), we know that the next entry on STARTLIST points to the current sub-formula if and only if GRECCNT is equal to the bottom third of the next entry on STARTLIST. GRECCNT is set to zero before PPR calls PPR1 and before PPR calls PPR2.

ENDLIST is merely the last cell of STARTLIST. It is used for fast NCONCing.

REMAINDER is a number. It represents the number of spaces FMLA can be shoved over towards the right after pretty-printing it. It is from REMAINDER that we learn whether or not indentation of the best kind is possible, and, if not, how much indentation of the other kind is possible.

The Locals of PPR1

PPR1 has a number of arguments and local variables.

The arguments are FMLA and RPARCNT.

FMLA is the current sub-formula we are working on.

RPARCNT is the number of characters we want to output on the last line after pretty-printing FMLA. Supposing that the user has not requested any, then RPARCNT is simply the number of right parentheses that is to be printed on the last line after pretty-printing FMLA, i.e., right parentheses belonging to formulas of which FMLA is a sub-formula that must be printed on the same line immediately after FMLA. Any pretty-print algorithm which ignores this aspect of pretty-printing is, in my opinion, not exact. The secret to keeping track of the right parentheses is to call PPR1 recursively with RPARCNT as 0 unless you are recursing on the last member of FMLA, in which case you call PPR1 recursively with RPARCNT as RPARCNT + 1. When eventually the last member of FMLA is an atom, simply add in RPARCNT when calculating the space it will take to print the atom. (Actually RPARCNT is one greater than suggested here.)

A global variable called SPACELEFT ought really to be considered as an argument of PPR1. SPACELEFT is decremented by one every time we recurse into PPR1 and incremented by one every time we exit. Hence it does not have to be local, and we have not made it local. SPACELEFT is initialized to MARG2-MARG1 by PPR before it calls PPR1. Hence SPACELEFT represents the maximum width of characters that can be available for pretty-printing FMLA. That is because with minimum indenting we will space at least once for the arguments of every formula of which FMLA is a sub-formula.

The local variables of PPR1 are:

NODENAME is set to the value of GRECCNT on entry to PPR1. Then GRECCNT is incremented by one. NODENAME is to be thought of as pointing to FMLA.

RUNFLAT is a variable in which we accumulate the flatsize of FMLA (the number of characters it takes to print FMLA flat).

MINREM is a variable in which we keep the minimum amount of space remaining after pretty-printing each of the arguments of FMLA. MINREM is basically just the smallest value REMAINDER has had after recursive calls of PPR1 on the arguments of FMLA.

L is just a variable that is used to CDR down FMLA as we call PPR1 on each of the arguments of FMLA.

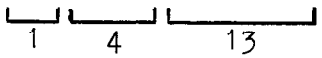
RUNSTART is a variable used to concatenate the STARTLIST's which are returned by recursive calls of PPR1 on the arguments of FMLA. RUNEND is used to help NCONC together different STARTLIST's. RUNEND is always the last cell of RUNSTART.

The Flow of Control through PPR1

After initialization and assuming FMLA does not have length 1, we begin looping around LOOPFLAT, CDRing down the arguments of FMLA. The tentative assumption is that FMLA can be printed flat. As long as each of the arguments of FMLA can be printed flat, we simply keep summing up the FLATSIZE's in RUNFLAT and keep MINREM to be the least of the REMAINDER's.

If we finish CDRing down FMLA and each argument will fit flat, we ask if RUNFLAT is $\leq$ SPACELEFT. If so, then we can print FMLA flat.

If not, we set FLATSIZE to false and set STARTLIST to the list whose only element is a packed number of the form:



The top bit is set if the best kind of indenting is possible. The 4 bits are set to the amount we further indent the arguments. The bottom 13 bits are set to NODENAME. These packed numbers are set up by PPRPACK.

If, while CDRing down FMLA, we discover that some argument requires indentation (FLATSIZE is false), then we know that the arguments of FMLA must be indented also.

We initialize RUNSTART to be the STARTLIST defined by the first argument which requires indenting, and then we begin looping around LOOPIND. Here we call PPR1 recursively on each of the remaining arguments to FMLA. If FLATSIZE is false after any of these calls, we concatenate the STARTLIST defined by that call to the end of RUNSTART.

When we reach NIL, we cons the packed number (as above) onto RUNSTART and assign that to STARTLIST, and set FLATSIZE to false.

Following is LISP code for PPR, PPR1, PPR2, and PPRPACK. There are two functions used but not defined:

PPRDL takes one atomic argument and returns the number of characters in it.

PPRSP takes one numeric argument and prints that many spaces.

Finally, there is an example of the kind of formula for which PPR was written. It is printed at various MARG2 settings.

I thank J Moore for his substantial help in the coding, debugging and documenting of PPR.

HERE IS THE LISP CODE FOR PPR.

```
(DEFPROP PPR
         (LAMBDA (FMLA MARG1 RPARCNT)
                 (PROG NIL
                       (COND ((ATOM FMLA) (PRIN1 FMLA) (RETURN NIL)))
                       (SETQ GRECCNT 0)
                       (SETQ SPACELEFT (DIFFERENCE MARG2 MARG1))
                       (PPR1 FMLA (ADD1 RPARCNT))
                       (COND (FLATSIZE (PRINT FMLA) (RETURN NIL)))
                       (SETQ NEXTNODE (LOGAND (CAR STARTLIST) 8191))
                       (SETQ NEXTIND (LOGSHIFT (CAR STARTLIST) (MINUS 13)))
                       (SETQ GRECCNT 0)
                       (PPR2 FMLA MARG1)))
         EXPR)


(DEFPROP
 PPR1
 (LAMBDA
  (FMLA RPARCNT)
  (PROG
   (NODENAME DLHDFMLA RUNFLAT MINREM L RUNSTART RUNEND)
   (SETQ NODENAME GRECCNT)
   (SETQ GRECCNT (ADD1 GRECCNT))
   (SETQ DLHDFMLA (ADD1 (PPRDL (CAR FMLA))))
   (COND ((NULL (CDR FMLA))
          (SETQ FLATSIZE (PLUS RPARCNT DLHDFMLA))
          (SETQ REMAINDER (DIFFERENCE SPACELEFT FLATSIZE))
          (RETURN NIL)))
   (SETQ RUNFLAT DLHDFMLA)
   (SETQ MINREM (DIFFERENCE SPACELEFT DLHDFMLA))
   (SETQ SPACELEFT (SUB1 SPACELEFT))
   (SETQ L FMLA)
  LOOPFLAT
   (SETQ L (CDR L))
   (COND ((NULL L)
          (SETQ SPACELEFT (ADD1 SPACELEFT))
          (COND ((OR (EQUAL RUNFLAT SPACELEFT) (LESSP RUNFLAT SPACELEFT))
                 (SETQ FLATSIZE RUNFLAT)
                 (SETQ REMAINDER (DIFFERENCE SPACELEFT RUNFLAT)))
                (T (SETQ STARTLIST (CONS (PPRPACK) NIL))
                   (SETQ ENDLIST STARTLIST)
                   (SETQ FLATSIZE NIL)))
          (RETURN NIL)))
   (COND ((ATOM (CAR L))
          (SETQ TEMP1 (PPRDL (CAR L)))
          (SETQ RUNFLAT (PLUS (ADD1 TEMP1) RUNFLAT))
          (SETQ TEMP1 (DIFFERENCE SPACELEFT TEMP1))
          (COND ((NULL (CDR L))
                 (SETQ RUNFLAT (PLUS RPARCNT RUNFLAT))
                 (SETQ TEMP1 (DIFFERENCE TEMP1 RPARCNT))))
```

```
          (COND ((LESSP TEMP1 MINREM) (SETQ MINREM TEMP1)))
          (GO LOOPFLAT)))
   (PPR1 (CAR L) (COND ((CDR L) 1) (T (ADD1 RPARCNT))))
   (COND ((LESSP REMAINDER MINREM) (SETQ MINREM REMAINDER)))
   (COND
       (FLATSIZE (SETQ RUNFLAT (PLUS (ADD1 FLATSIZE) RUNFLAT)) (GO LOOPFLAT)))
   (SETQ RUNSTART STARTLIST)
   (SETQ RUNEND ENDLIST)
  LOOPIND
   (SETQ L (CDR L))
   (COND ((NULL L)
          (SETQ STARTLIST (CONS (PPRPACK) RUNSTART))
          (SETQ ENDLIST RUNEND)
          (SETQ FLATSIZE NIL)
          (SETQ SPACELEFT (ADD1 SPACELEFT))
          (RETURN NIL)))
   (COND ((ATOM (CAR L))
          (SETQ TEMP1 (DIFFERENCE SPACELEFT (PPRDL (CAR L))))
          (COND ((NULL (CDR L)) (SETQ TEMP1 (DIFFERENCE TEMP1 RPARCNT))))
          (COND ((LESSP TEMP1 MINREM) (SETQ MINREM TEMP1)))
          (GO LOOPIND)))
   (PPR1 (CAR L) (COND ((CDR L) 1) (T (ADD1 RPARCNT))))
   (COND ((LESSP REMAINDER MINREM) (SETQ MINREM REMAINDER)))
   (COND (FLATSIZE) (T (RPLACD RUNEND STARTLIST) (SETQ RUNEND ENDLIST)))
   (GO LOOPIND)))
 EXPR)



(DEFPROP
 PPR2
 (LAMBDA
  (FMLA MARG1)
  (PROG
   (NONLFLAG INDFLAG)
   (COND ((ATOM FMLA) (PRIN1 FMLA) (RETURN NIL)))
   (COND
     ((EQUAL GRECCNT NEXTNODE)
      (SETQ MARG1 (PLUS MARG1 (LOGAND NEXTIND 15)))
      (SETQ INDFLAG T)
      (SETQ NONLFLAG (EQUAL 16 (LOGAND NEXTIND 16)))
      (SETQ STARTLIST (CDR STARTLIST))
      (COND (STARTLIST (SETQ NEXTNODE (LOGAND (CAR STARTLIST) 8191))
                       (SETQ NEXTIND (LOGSHIFT (CAR STARTLIST) (MINUS 13))))))
     (T (SETQ INDFLAG NIL) (SETQ NONLFLAG T)))
   (SETQ GRECCNT (ADD1 GRECCNT))
   (CUCHAROUT 24)
   (PRIN1 (CAR FMLA))
   (SETQ FMLA (CDR FMLA))
   (COND ((NULL FMLA) (CUCHAROUT 25) (RETURN NIL)))
   (COND (NONLFLAG (CUCHAROUT 16)) (T (CUCHAROUT 17) (PPRSP MARG1)))
  LOOP
   (PPR2 (CAR FMLA) MARG1)
   (SETQ FMLA (CDR FMLA))
   (COND ((NULL FMLA) (CUCHAROUT 25) (RETURN NIL)))
   (COND (INDFLAG (CUCHAROUT 17) (PPRSP MARG1)) (T (CUCHAROUT 16)))
   (GO LOOP)))
 EXPR)
```

```
(DEFPROP
 PPRPACK
 (LAMBDA
  NIL
  (LOGOR
   (LOGSHIFT
    (COND
          ((LESSP MINREM DLHDFMLA) (SETQ REMAINDER 0) (ADD1 MINREM))
          (T (SETQ REMAINDER (DIFFERENCE MINREM DLHDFMLA)) (PLUS 17 DLHDFMLA)))
    13)
   NODENAME))
 EXPR)
```

HERE IS A TYPICAL FORMULA FROM OUR THEOREM PROVER PRETTY PRINTED WITH MARGS SET TO 50.

```
(IF
 A
 (IF
  (EQUAL (LAST GENRL1) (CAR A))
  (IF
   (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1).
   (IF
    A
    (IF
     GENRL1
     (IF
      (EQUAL (LAST GENRL1) (CAR A))
      (IF
       (APPEND GENRL1 (CONS A1 NI_))
       (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))).
              A1)
       (EQUAL GENRL11 A1))
      T)
     (IF
      (EQUAL GENRL11 (CAR A))
      (IF
       (APPEND GENRL1 (CONS A1 NI_))
       (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))).
              A1)
       (EQUAL GENRL11 A1))
      T))
    T)
   T)
  (IF
   A
   (IF
    GENRL1
    (IF
     (EQUAL (LAST GENRL1) (CAR A))
     (IF
       (APPEND GENRL1 (CONS A1 NI_))
       (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))).
              A1)
       (EQUAL GENRL11 A1))
     T)
    (IF
     (EQUAL GENRL11 (CAR A))
     (IF
       (APPEND GENRL1 (CONS A1 NI_))
       (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))).
              A1)
       (EQUAL GENRL11 A1))
     T))
   T))
 (IF
  A
  (IF
   GENRL1
```

```
 (IF
  (EQUAL (LAST GENRL1) (CAR A))
  (IF
     (APPEND GENRL1 (CONS A1 NIL))
     (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL)))
            A1)
     (EQUAL GENRL11 A1))
  T)
 (IF
  (EQUAL GENRL11 (CAR A))
  (IF
     (APPEND GENRL1 (CONS A1 NIL))
     (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL)))
            A1)
     (EQUAL GENRL11 A1))
  T))
T))
```

THIS IS THE SAME FORMULA PRINTED WITH MARG2 SET TO 65.

```
(IF
 A
 (IF
  (EQUAL (LAST GENRL1) (CAR A))
  (IF
   (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
   (IF
      A
      (IF GENRL1
          (IF (EQUAL (LAST GENRL1) (CAR A))
              (IF (APPEND GENRL1 (CONS A1 NIL))
                  (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                  (EQUAL GENRL11 A1))
              T)
          (IF (EQUAL GENRL11 (CAR A))
              (IF (APPEND GENRL1 (CONS A1 NIL))
                  (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                  (EQUAL GENRL11 A1))
              T))
      T)
   T)
  (IF A
      (IF GENRL1
          (IF (EQUAL (LAST GENRL1) (CAR A))
              (IF (APPEND GENRL1 (CONS A1 NIL))
                  (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                  (EQUAL GENRL11 A1))
              T)
          (IF (EQUAL GENRL11 (CAR A))
              (IF (APPEND GENRL1 (CONS A1 NIL))
                  (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                  (EQUAL GENRL11 A1))
              T))
      T))
 (IF A
     (IF GENRL1
         (IF (EQUAL (LAST GENRL1) (CAR A))
             (IF (APPEND GENRL1 (CONS A1 NIL))
                 (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                 (EQUAL GENRL11 A1))
             T)
         (IF (EQUAL GENRL11 (CAR A))
             (IF (APPEND GENRL1 (CONS A1 NIL))
                 (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                 (EQUAL GENRL11 A1))
             T))
     T))
```

FINALLY, HERE IS THE SAME FORMULA PRINTED WITH MARG2 SET TO 78.

```
(IF A
    (IF (EQUAL (LAST GENRL1) (CAR A))
        (IF (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
            (IF A
                (IF GENRL1
                    (IF (EQUAL (LAST GENRL1) (CAR A))
                        (IF (APPEND GENRL1 (CONS A1 NIL))
                            (EQUA_ (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                            (EQUA_ GENRL11 A1))
                        T)
                    (IF (EQUAL GENRL11 (CAR A))
                        (IF (APPEND GENRL1 (CONS A1 NIL))
                            (EQUA_ (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                            (EQUA_ GENRL11 A1))
                        T))
                T)
            T)
        (IF A
            (IF GENRL1
                (IF (EQUAL (LAST GENRL1) (CAR A))
                    (IF (APPEND GENRL1 (CONS A1 NIL))
                        (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                        (EQUAL GENRL11 A1))
                    T)
                (IF (EQUAL GENRL11 (CAR A))
                    (IF (APPEND GENRL1 (CONS A1 NIL))
                        (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                        (EQUAL GENRL11 A1))
                    T))
            T))
    (IF A
        (IF GENRL1
            (IF (EQUAL (LAST GENR_1) (CAR A))
                (IF (APPEND GENRL1 (CONS A1 NIL))
                    (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                    (EQUAL GENRL11 A1))
                T)
            (IF (EQUAL GENRL11 (CAR A))
                (IF (APPEND GENRL1 (CONS A1 NIL))
                    (EQUAL (LAST (APPEND GENRL1 (CONS A1 NIL))) A1)
                    (EQUAL GENRL11 A1))
                T))
        T))
```