1986

# Language Independent PROTRAN

John R. Rice
*Purdue University*, jrr@cs.purdue.edu

Report Number:
86-633

# LANGUAGE INDEPENDENT PROTRAN

John R. Rice*
Computer Science
Purdue University

CSD-TR 633
October 6, 1986

## Abstract

The objective of this paper is to explore the possibility that PROTRAN can be made essentially language independent. PROTRAN currently exists as a system to provide a high level interface through Fortran to the problem solving capabilities of the IMSL library. We claim that an extension can be made to provide such interfaces to almost all interesting languages using the existing PROTRAN interface, much of the existing PROTRAN language processing software and the existing software parts.

* Work supported in part by the Software Engineering Research Center

# LANGUAGE INDEPENDENT PROTRAN

## 1. INTRODUCTION AND SUMMARY

PROTRAN is a system that provides a high level interface through Fortran to problem solving capabilities from the IMSL library of routines. Its two principal objectives are:

Ease of use

Reliable use (checking of problem formulation)

It is an example of an approach to a software parts technology in that the PROTRAN interface is high level, natural and flexible so that underlying software can be changed, enhanced, etc. without perturbing the user code.

For the sake of brevity, we assume that the reader has some familiarity with PROTRAN. The book [Rice, 1983] is the most accessible description, especially in Chapter 15. The paper [Aird and Rice, 1983] describes the design and implementation technically and the details of PROTRAN's use is given the manuals [IMSL, 1985, 1986].

The objective of this paper is to explore the possibility that PROTRAN can be made essentially language independent. That is, most existing languages can be extended by PROTRAN problem solving statements. We claim that the extension can be made using the existing PROTRAN framework, much of the existing PROTRAN language processing software and the existing software parts (Fortran routines).

## 2. AN ABSTRACT VIEW OF PROTRAN

PROTRAN is a facility that provides problem solving extensions to languages in a general, "almost" language independent way. There are three ingredients to PROTRAN:

1. *Communication of Variables.* This is a correspondence between some PROTRAN variables and some language variables with a mechanism for carrying out the correspondence. The

data types and variables of PROTRAN and the language overlap, but neither needs to include the other.

2. *PROTRAN Declarations.* These are statements that declare PROTRAN variables.

3. *PROTRAN Problem Solving Statements.* These are PROTRAN statements interspersed in the language which "solve problems" and create new values for PROTRAN and/or language variables.

To illustrate this view, we give an example extension to the Basic language which provide a differential equation solving capability. This is shown in Figure 1. The PROTRAN statements include the word PROTRAN so it is obvious where the extension are made. The Basic program has only real (scalar) data types while the PROTRAN extension has real scalar and function data types. There are variables with corresponding types and names (N, U1, E, A1, A2 and A3) which will have the same values.

It is the thesis of this paper that the simple idea illustrated in Figure 1 can be extended to a wide class of important languages and problem solving extensions. In the next two sections we outline how this may be done in more detail.

## 3. PROCESS VIEW OF PROTRAN

PROTRAN is based on a two pass approach to language processing. The input is an application language (which we call APLA for short) extended using PROTRAN. The PROTRAN preprocessor reads this input and produces code in APLA and a base language (which we call Fortran because that is what would be used in any prototype of a language independent PROTRAN). The APLA code would be a "normal" APLA program which would execute and produce the desired results. Within this program would be "escapes" to the base language where the problem solving takes place. Note that the two pass approach is not essential to the concept, it is a way to provide extensions without perturbing the underlying language processor (APLA

```
10  LET U1 = 6.
20  LET E = 0.0001
30  PROTRAN DECLARE
40              FUNCTION  F, G
50              REAL      N, U1, E, X, Y, T, U, A1, A2, A3
60  PROTRAN END DECLARE
70  READ N
80  LET  U1 = U1/N
90  PROTRAN ASSIGN F(X, Y) = X**2 + Y**2 + N*SIN(X*A)
100 READ A
110 PROTRAN SOLVE U' = F(U, T)
120             FOR (T = 0, N)
130             ANSWER = G(T)
140             INITIAL = U1/2 + 2.1*N
150             ERRTARGET = E
160 PROTRAN END SOLVE
170 PROTRAN ASSIGN A1 = G(1); A2 = G(2); A3 = G(N)
180 LET  B = A1 + A2
190 LET  C = A2 + A3
200 PRINT A, B, C
210 STOP
```

**Figure 1.** A hypothetical PROTRAN extension of Basic to solve ordinary differential equations. The variables in Basic of this program are U1, E, N, A, A1, A2, A3, B and C.

compiler).

Examples of such escapes are

1. *Fortran procedures are callable from APLA*:

```
CALL PROB13(IN1, IN2, IN3, ..., INk, OUT1, OUT2, ..., OUTm)
```

Here the Fortran code receives all the input variables from APLA and maps them into Fortran representations. It uses this plus its own variables (derived from the PROTRAN statements) to solve the problem. It then converts the appropriate variables back to APLA form and returns these through the procedure interface.

2. *APLA has an interactive interface*

PRINT OUT1, OUT2, ..., OUTk

READ IN1, IN2, ..., INm

The variables are all the same as the previous example, but the names are switched between IN and OUT. The I/O file assignments must be set such that this PRINT file goes to the Fortran process (which has been suspended waiting on input). When the problem is solved the Fortran process provides input for the APLA READ statement (which has been suspended waiting on input).

There are other escape mechanisms that one can use, but these two represent the extremes between high efficiency (such as Fortran and assembler) and low efficiency. The PRINT/READ mechanisms can be used even when APLA and Fortran do not run on the same machine. It is essential that APLA have some escape to the base language.

From the process point of view, we see that PROTRAN has the following ingredients:

1. *A preprocessor.* This creates the executable APLA program and the auxiliary PROTRAN (Fortran) program.

2. *An escape mechanism.* This allows one to transfer control and data between the two languages. Good efficiency is, of course, desirable here.

3. *Data type mappings.* Procedures must be written to map data types between APLA and the base language. Some of these are trivial (e.g., for numbers or simple arrays), some of these are difficult (e.g., expressions or records) and some are impossible or more accurately, too difficult to be worthwhile (e.g., representation of a complex object in a 3D geometric modeler).

4. *Remote procedure execution.* We obviously have a form of remote procedure execution when APLA calls on Fortran to solve a particular problem. Less obvious but still important instances arise when data type mappings are too difficult. For example, suppose one wants to return to a Basic program the function which has been created to solve a problem (e.g., the function $g(x)$ in Figure 1). It is probably more practical to have the Basic program execute the For-

tran function when it needs a value. There will be similar situations when the data to the problem solving statement involves a substantial amount of APLA code (not just simple variables or expressions). Then remote procedure execution from Fortran is accomplished using the reverse communication idea.

We observe that these four ingredients can be created for very wide ranges of languages and applications. Surely in some instances there will be serious efficiency penalties or even inherent algorithmic problems. Recall that one key assumption is that the PROTRAN preprocessor is to do only trivial analysis of the input programs (e.g., it must know where it ends).

## 4. IMPLEMENTATION VIEW OF PROTRAN

The current PROTRAN has Fortran as both the base and application language. Thus it is natural for it to merge the output of the preprocessor into a single Fortran program. Much efficiency and internal consistency is thus gained.

The current PROTRAN preprocessor is written by a *preprocessor generator* system and is in Fortran. This preprocessor generator is quite flexible and it would be relatively easy to change the superficial form of the PROTRAN statements to be compatible with other application languages. We discuss this more later.

A key concern that arises when the base and application languages are separated is to maintain a consistent view of the state of the computation. Since the PROTRAN approach is to interact very little with the APLA program, most of the burden must be placed on the Fortran program PROTRAN generates. It must create symbol tables for all its variables and keep those "current" which are shared with the APLA program. Furthermore, the Fortran program must reflect the scope rules of the APLA program. Thus, the PROTRAN preprocessor must analyse the input program enough so as to determine the scope as needed. One can visualize that this could create tricky problems for some situations.

We propose that the current syntactic structure of PROTRAN be kept in any language independent versions. We summarize that structure as follows:

1. *Declarations*. The declaration has three parts.

**head:** A keyword indicating that PROTRAN declarations are starting.

**body:** A list of declarations of the general form (or its transpose).

<data type> <list of variables>

**close:** A keyword indicating the end of a PROTRAN statement.

The body is read only by the PROTRAN preprocessor so it can have any form desired. However, there are certain styles to declarations in various languages and it makes sense for the PROTRAN versions to choose styles compatible with APLA.

2. *Problem Solving*. These executable statements have four parts.

**head:** Keyword identifying the problem solving statement.

**problem:** A form which naturally expresses the problem to be solved. It may vary widely in construction for different problem domains.

**arguments:** An unordered list of "phrases" of the form:

<keyword> <data>

Various punctuations may be used to make the phrases appear natural.

**close:** A keyword indicating the end of a PROTRAN statement. Note that APLA variables may enter the problem solving statements through either the *problem* or *arguments* parts.

We expect that only cosmetic changes would be needed to adapt most PROTRAN preprocessors from one application language to another as far as accommodating different styles and syntax in the statements.

In Figure 2 we give a sample of PROTRAN statements from the existing PROTRAN systems (Math PROTRAN, LP PROTRAN), see [IMSL, 1985, 1986]. In Figure 3 we give a sample of potential PROTRAN statements in other problem domains and with different language styles. These are illustrative in nature and longer than normal keywords are used to make the statements more self explanatory. A wide range of mathematical areas are currently appropriate for PROTRAN statements including the following areas in the undergraduate Math/CS curriculum:

college algebra (symbolic)

trigonometry (symbolic)

calculus (symbolic and numerical)

linear algebra (symbolic and numerical)

vector calculus (symbolic and numerical)

ordinary differential equations (numerical)

coordinate transforms (symbolic and numerical)

linear programming

statistics

sorting/manipulating lists, records, files

creating data structures

graphics

pattern matching

parsing from gramars

```
$ DECLARATIONS
    VECTOR CHEBY(11), FX(11), Y(4), F(4), YSTART(4)
    MATRIX TABLE Y(50, 4)
$ ASSIGN  CHEBY(J) = -10.* COS(2*J+1)*PI/20)
          FX(J) = F(CHEBY(J))
$ INTERROLATE, FX
       USING POLYNOMIALS;  BY POLY_FX;  VS CHEBY
$ MAX  ABS(POLY_F(X) - F(X));    FOR (X = -5., 5., .05)
       IS ERRMAX
$ PLOT,  POLY_F(X);  FOR (X = -5, 5)
       TITLE = "Polynomial interpolation on Chebysher points for 1/(1 + x*x)"
       XLABEL = "X-Axis";  YLABEL = "Poly. value"

$ DIFEQN  Y' = F(T, Y)
          ON (1.0, 10.0)    ;    IINITIAL = YSTART
          ERRTARGET = .0001
          NOUTPUT = 91;    SOLUTION = TABLEY
          DEFINE
          = = = =
            F(1) = Y(1) - T*Y(2) + SIN(T*Y(3))
            F(2) = Y(2)*Y(3) - COS(T*Y(1) - Y(4))
            F(3) = EXP(-Y(3)*T)*T*Y(4)
            F(4) = COS(Y(1)*Y(4)*T) - SIN(Y(2)*Y(3)*T)
          = = = =
$ PRINT TABLEY

$ LP PROBLEM ;    OBJECTIVE
          = = = =
            2*X + 6*Y
          = = = =
     CONSTRAINTS
          = = = =
            2*X + Y   .LE.  4
            -X + 3*Y    .LE. -2
          = = = =
$ LP MAXIMIZE ;    OBJECTIVE = Salary
                   PRIMAL = Variables
                   TITLE = "Salary maximization problem"
```

**Figure 2.** Example problem solving statements from math PROTRAN,and LP PROTRAN.

```
DERIVATIVE   (x^2 + (ax)^(x-1))/(1. + ae^((ax-b)^2/(1+x)))max(3B + 7.2, x^2 - 3)
             VARIABLE = x;   ORDER = 3;   IS F3(x)
             VALUE_AT ZPTS = F3_ZPTS

POLYNOMIAL   (x - y + z)^3(x^2 - 3xz + 2x - 4) + (6x - 4y + z^2)^3(x^3y^2 - 3xz + 2y - 3)
             BY_POWERS_OF x;   IS P4(x)
             COEFS = P4_COEF;   DEGREE = P4_DEGREE

CHANGE VARIABLE   (3A + x)f.x.x - (B + y)f.y.y + cos(x + y)f.x - (x^2 + y)f.y
                  RECT_TO_POLAR (x, y) → (R, T)
                  IS New_Diff_Eq.

SIMPLIFY   Expression_4
           USING x^2 + y^2 = 1;   USING sin^2(x) + cos^2(x) = 1.
           USING B = 0;   USING A = 2

POWER_SERIES   FCN3(x, y, t);   AT   (x, y, t) = (1., -3., T4)
               TERMS (x = 20), (y = 12), (t = 4)
               COEFFICIENTS = FCN3_series

FILES   MEM.REC [Name, Title, Ph.Home, Ph.Office, ZIP, years]
        EXTRACT Title = Dr.;   EXTRACT ZIP = [0, 31999]
        SORT_ON ZIP, Ph.Office
        IS MEM.REC.MD.EAST (ZIP, Ph.Office, Name)

PATTERN   MEM.REC (Name, Title, Ph.Home, Ph.Office, ZIP, years)
          PATTERN(1)  Name.last = Jones,
                          ZIP in 9*
                          years in [3–8]
                      IS JONES_list
          PATTERN(2)  Title = President or *Chairman* or *Secretary*
                          years > 10
                          ZIP in [0, -3*]
                      IS Big.shot (Name, Ph.Office, Title)
                      SORT_ON Name

PLOT   f(t), g(t), z(t) = |f(t) - g(t)|/|f(t)| ; ON t = [0, 1]
       COLOR f = red, g = blue, z = green
       LABEL "original curve", red, LINE
             "estimated behavior", blue, LINE
             "relative difference", green, LINE
```

**Figure 3.** Some hypothetical PROTRAN statements.

## REFERENCES

T.J. Aird and J.R. Rice, PROTRAN: Problem solving software, *Adv. Engineering Software*, 5 (1983), 202–206.

IMSL, Inc., Math PROTRAN Users Manual (1985)

IMSL, Inc., Stat PROTRAN Users Manual (1985)

IMSL, Inc., LP PROTRAN Users Manual (1986)

J.R. Rice, *Numerical Methods, Software and Analysis*, McGraw-Hill, New York, (1983)