DECUS
PROGRAM LIBRARY

DECUS NO.	8-526
TITLE	PROCAL 10/71
AUTHOR	Peter G. Kretzman
COMPANY	Cold Spring Harbor High School Huntington, New York
DATE	March 31, 1972
SOURCE LANGUAGE	PAL-D

PROCAL 10/71

DECUS Program Library Write-up DECUS NO. 8-526

NAME: PROCAL 10/71, an acronym for PROpositional CALculus.

ABSTRACT: PROCAL 10/71 is a conversational symbolic logic programming system that uses the logical notation developed by J. Lukasiewicz of Poland, and is especially oriented towards solving logical problems. Using the special notation, the user can input any number of assumed logical premises, and then see if his proposed logical conclusion is valid or invalid, based upon those premises.

STORAGE: Main program from 0-1777; rest of memory serves as buffers and pushdown.

LOGICAL THEORY:

In any sort of logical problem such as the problem concerning the 'Tardy Bus' in Appendix A, there are five major ways of connecting "true or false" types of sentences. These connectors are called:

(1) Implication

(2) Disjunction

(3) Conjunction

(4) Equivalence

(5) Negation

For purposes of simplification, we will use these two sentences for demonstration of these five types of logical connectors:

(1) John goes to the movies.

(2) Mary goes to the movies.

It will be seen that there are many ways of combining these two sentences, but the primary ways are the five listed above.

IMPLICATION:

IF John goes to the movies, THEN Mary goes to the movies.

We will now set up a truth table, which is a true-false representation of all the possible combinations of the two original sentences. To do this, we will assign variable names to these sentences. These variables, therefore may have either of two possible values-'true' or 'false'. We will let 'P' represent John going, and 'Q' represent Mary going. The truth table will therefore look like this:

P	Q	IF P, THEN Q (represented by "P→Q")
T	T	T
T	F	F
F	T	T
FF	F	T

T-TRUE
F_FALSE

Therefore, the only case where P→Q (read as "if P, then Q") is false is when P is true and Q is false.

DISJUNCTION:

John goes to the movies OR Mary goes to the movies.

P	Q	P OR Q (represented by "P v Q")
T	T	T
T	F	T
F	T	T
F	F	F

T-TRUE
F-FALSE

Therefore, the only case where P OR Q is false is when neither P nor Q is true. (For you logicians, this is an inclusive or)

CONJUNCTION:

John goes to the movies AND Mary goes to the movies.

P	Q	P AND Q (represented by "P $\wedge$ Q")
T	T	T
T	F	F
F	T	F
F	F	F

T-TRUE
F-False

Therefore, "P AND Q" is true only when P is true and Q is also true.

EQUIVALENCE:

John goes to the movies IF, AND ONLY IF, Mary goes to the movies.

P	Q	P IF, AND ONLY IF, Q (represented by "P $\equiv$ Q")
T	T	T
T	F	F
F	T	F
F	F	F

T-TRUE
F-FALSE

Therefore, for "P $\equiv$ Q" to be true, P must have the same truth value as Q.

NEGATION:

John does NOT go to the movies.

P	Q	NOT P	NOT Q
T	T	F	F
T	F	F	T
F	T	T	F
F	F	T	T

Therefore, Negation operates on only one variable, instead of two as in the others, and is merely a reversal of the variables truth value.

PROCAL NOTATION:

Procal notation is unique from standard notation (which I have just covered) in the fact that it needs no parenthesis or unusual characters such as $\rightarrow$ or $\wedge$. It also, however has all of these five basic types of logical connectives. To transfer a standard logical expression to PROCAL notation, one may merely attach a prefix to the logical variables. Here are the assigned prefixes:

IMPLICATION:	C	$\rightarrow$
DISJUNCTION:	A	$\vee$
CONJUNCTION:	K	$\wedge$
EQUIVALENCE:	E	$\equiv$
NEGATION:	N	$\sim$

Therefore, to encode this standard expression in PROCAL notation:

$$(P \rightarrow Q) \equiv \sim(P \wedge \sim Q)$$

First encode anything in parenthesis, then combine together:

(1) $P \rightarrow Q$: CPQ

(2) $P \wedge \sim Q$: KPNQ

(3) $\sim(P \wedge \sim Q)$: NKPNQ

(4) $(P \rightarrow Q) \equiv \sim(P \wedge \sim Q)$: ECPQNKPNQ

For more examples of this method of coding, see Appendix A and the TARDY BUS PROBLEM.

OPERATION:

PROCAL 10/71 is loaded into core using the standard DEC binary loader. PROCAL is also compatible with the 4K Disk Monitor System, and the dialogue with the system would look like this:

.LOAD↵
*IN-T:↵
*
ST=7600↵
↑↑
.SAVE PROCAL!0-1777; 1000↵
.

Starting address is 1000 octal.

When first started, PROCAL will print a heading with the name, author, and version number of the program. It will then proceed to ask for a logical premise by typing:

PREMISE #1:

The user may now input his premise, with the special notation previously explained. When a carriage return is struck, PROCAL will check the logical expression or WFF (Well Formed Formula) for errors. The three errors it checks for at this stage are:

1) Illegal character (not P,Q,R,S,T,U,C,A,K,E,N)
2) A WFF beginning with C,A,K or E (CAKE-WFF) does not have two logical expressions following.
3) A WFF beginning with N (N=WFF) does not have one logical expression following.

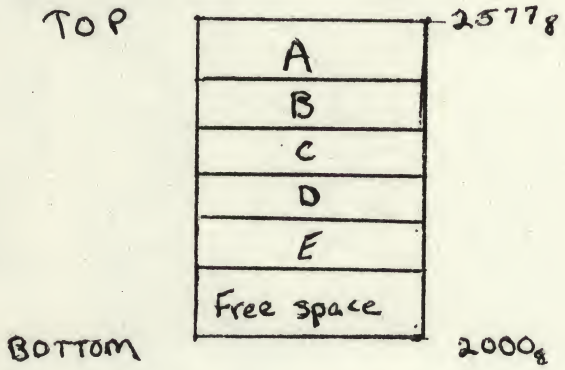
If any of these errors are picked up, PRROCAL will type "ILLOGICAL", and ask for the premise again. When a logically sound WFF has finally been input, PROCAL will store it, and ask for another, in the same fashion. When the user finishes putting in the premises, he should type ALTMODE as the first character of a line. (ALTMODE is also labeled PREFIX OR ESCAPE) PROCAL will now request a logical conclusion that supposedly follows from the previously inputted set of premises. The same logical errors are checked for with the conclusion as with the premises. However, the user must make sure to use his logical variables in strict order, for if one needs two variables, and chooses P and T, PROCAL will determine that there is a SX (syntax) error. When a carriage return is struck on the conclusion, and PROCAL detects no errors, the logical entity of premises and conclusion is evaluated, and PROCAL will then type whether the conclusion is valid or invalid, based upon that set of premises. It is then possible to check any number of conclusions on that set of premises, as PROCAL will then ask for another conclusion, wiping the first from its memory. If it desired to return to the premise mode, and re-input the premises, the user may type ALTMODE. At any time of input, a control C will return the user to the core resident monitor.
NOTICE: A WFF in PROCAL must be less than 63 characters long, due to limited space.

THEORY:

PROCAL 10/71's main algorithm is the use of a binary truth table mathematically set up in core memory for evaluation. This truth table resides from 2000-2577. This is where PROCAL obtains the truth value for the variables in a given instance. This truth table covers all the possible combinations for

the number of variables currently in the premises and conclusion combined.

Procal's second major algorithm is the use of a recursive subroutine, coupled with several pushdown stacks. Normally, a machine language subroutine may not be reentered from within itself, since the return address is stored as the first location of the subroutine and would thus be overwritten upon duplicate entry. However, if some way were found to store that return address, along with any temporary and intermediate subroutine data accumulated in that run in a storage area, it would be possible to reenter. Such a storage area is a pushdown stack (PDL), also referred to as a "first in, last out" stack. This pushdown stack has two operations: PUSHing and POPping, which are inverses of each other. Pushing is defined as placing a data word ON the top of the PDL and POPping is defined as taking a word OFF the top of the PDL. Here is a simplified diagram of a PDL:



The order in which these words were PUSHed was E,D,C,B,A. The order in which they will be POPped is A,B,C,D,E.

Therefore, using the PDL to store return addresses and temporary data, it is possible to call a subroutine recursively to any limit, as long as there is PDL space available.

This concept is the major reason for the existence of PROCAL, for it is used to evaluate the WFFs.

If the uses uses too many or too long WFFs, to an extent that there is no more core available lto the PDL, PROCAL will type "PUSHDOWN STACK OVERFLOW" and loop. PROCAL MAY NOT BE RESTARTED UNDER THESE CIRCUMSTANCES!!! However, this is

unlikely, since a great deal of room is allocated for pushdown.

REFERENCES:

Allen, Layman E. WFF 'n PROOF: The Game of Modern Logic
Autotelic Instructional Material Publishers, New Haven, Connecticut 1968

Copi, Irving M. Symbolic Logic, third edition, Macmillan, London 1967

Wrege, Doug. FOCAL: How to write New Subroutines and Use Internal Routines,
Atomic Energy Commission, 1968 (available from DECUS)

$

THE TARDY BUS PROBLEM

Given the following three statements as premises:

(1) If Bill takes the bus, then Bill misses his appointment, if the bus is late.

(2) Bill shouldn't go home, if (a) Bill misses his appointment, and (b) Bill feels downcast.

(3) If Bill doesn't get the job, then (a) Bill feels downcast, and (b) Bill should go home.

Is it valid to conclude:	YES	NO
Q1. That if Bill takes the bus, then Bill does get the job, if the bus is late?	____	____
Q2. That Bill does get the job, if (a) Bill misses his appointment, and (b) Bill should go home?	____	____
Q3. That if the bus is late, then (a) Bill doesn't take the bus, or Bill doesn't miss his appointment, if (b) Bill doesn't get the job?	____	____
Q4. That Bill doesn't take the bus, if (a) the bus is late, and (b) Bill doesn't get the job?	____	____
Q5. That if Bill doesn't miss his appointment, then (a) Bill shouldn't go home, and (b) Bill doesn't get the job?	____	____
Q6. That Bill feels downcast, if (a) the bus is late, or (b) Bill misses his appointment?	____	____
Q7. That if Bill does get the job, then (a) Bill doesn't feel downcast, or (b) Bill shouldn't go home?		
Q8. That if (a) Bill should go home, and Bill takes the bus, then (b) Bill doesn't feel downcast, if the bus is late?	____	____

CODING THE TARDY BUS PROBLEM

VARIABLE	MEANING
P	Bill takes the bus.
Q	Bill misses his appointment.
R	The bus is late.
S	Bill should go home.
T	Bill feels downcast.
U	Bill gets the job.

PREMISE	PROCAL Notation
1) $(P \wedge R) \rightarrow Q$	CKPRQ
2) $(Q \wedge T) \rightarrow {}^{N}S$	CKQTNS
3) ${}^{N}U \rightarrow (T \wedge S)$	CNUKTS

CONCLUSION	PROCAL Notation
1) $(P \wedge R) \rightarrow U$	CKPRU
2) $(Q \wedge S) \rightarrow U$	CKQSU
3) $R \rightarrow ({}^{N}P \vee ({}^{N}U \rightarrow Q))$	CRANPCNUQ
4) $(R \wedge {}^{N}U) \rightarrow {}^{N}P$	CKRNUNP
5) ${}^{N}Q \rightarrow ({}^{N}S \wedge {}^{N}U)$	CNQKNSNU
6) $(R \vee Q) \rightarrow T$	CARQT
7) $U \rightarrow ({}^{N}T \vee {}^{N}S)$	CUANTNS
8) $(S \wedge P) \rightarrow (R \rightarrow {}^{N}T)$	CKSPCRNT

PREMISE #1: CKPRQ
PREMISE #2: CKQTNS
PREMISE #3: CNUKTS
PREMISE #4:

CONCLUSION: CKPRU

VALID

CONCLUSION: CKQSU

VALID

CONCLUSION: CRANPCNUQ

VALID

CONCLUSION: CKRNUNP

VALID

CONCLUSION: CNQKNSNU

INVALID

CONCLUSION: CARQT

INVALID

CONCLUSION: CUANTNS

INVALID

CONCLUSION: CKSPCRNT

VALID

CONCLUSION:
*

A 3