```
                       *********************************
                       * PROLOG CROSS REFERENCE LISTING *
                       *********************************


                                The PRESS system


  PREDICATE                FILE           CALLED BY



&(2)                    invoca.pl     fixvar(2)

\=(2)                   miscel.pl

\\(2)                   invoca.pl

acollect(2)             invoca.pl     foreach(5)

acollect2(3)            invoca.pl     acollect(2) acollect2(3)

action(3)               iorout.pl     writefs(2)

acute(1)                simp

allempty(1)             multil.pl     mlmaplist(2)   mlmaplist(3)   mlmaplist(4)
                                      allempty(1)

andtodot(2)             press         maximum(2) andtodot(2)

angle(3)                goals         classify(2)

any(1)                  invoca.pl     any(1)

any1(3)                 simp          any1(3) collect(3) attract(3)

append(3)               listro.pl     append(3)  apply(2)  mlapply(2)  concat(3)
                                      splitperm(3) collect_ans(3)

apply(2)                applic.pl     checkand(2)     checklist(2)     mapand(3)
                                      maplist(3)  convlist(3) some(2) sublist(3)
                                      any1(3) rewrite(3) some(2)

arbint(1)               simp

arccoseval(2)           eval          eval1(2)

arcsineval(2)           eval          eval1(2) arctaneval(2)

arctaneval(2)           eval          eval1(2)

attract(3)              press         soll1(3)
```

attrax(4)	press	attract(3)
bag_routines(2)	simp	simplify2(2) tidy(2)
becomes(2)	assign.pl	tlim(1) gensym(2)
bigger(2)	press	smaller(2)
bind(3)	struct.pl	bind(3)
binding(2)	invoca.pl	
bloc(0)	goals	
blocineq(1)	goals	
cancelling(2)	simp	f12(2) reciprocal(2)
car(0)	goals	
casserta(1)	miscel.pl	simplify1(2)
cassertz(1)	miscel.pl	
cgensym(2)	miscel.pl	
changebag(4)	press	collect(3) attract(3)
changeunknown(4)	press	sol11(3)
checkand(2)	applic.pl	checkand(2)
checklist(2)	applic.pl	checklist(2) f13(2) eval(2) maximum1(2) run(0)
classify(2)	known	known(1)
clean(0)	miscel.pl	
cleanstack(0)	invoca.pl	dorun(2)
cnorm(2)	known	range(3)
collax(3)	press	collect(3)
collect(3)	press	sol11(3)
collect_ans(3)	press	findmax(3) collect_ans(3)
combine(3)	known	unionlist(2)
concat(3)	miscel.pl	gensym(2)
concavity(2)	goals	range(3)

```
conjineq(1)              goals

const(1)                 init          wordsin(2)

contains(2)              simp          freeofunks(1)    isolate(3)     collect(3)
                                       attract(3) changeunknown(4)

continue(0)              miscel.pl

convlist(3)              applic.pl     convlist(3)

cosean(1)                goals

coseval(2)               eval          eval1(2) taneval(2)

coueqn(1)                goals

cretract(1)              miscel.pl

cvalof(3)                assign.pl     gensym(2)

decomp(2)                simp          decomp(2)    bag_routines(2)    collect(3)
                                       attract(3) match(2) changebag(4)

diff(2)                  miscel.pl     changeunknown(4)

diffwrt(3)               press         findmax(3)

diffwrt1(3)              press         diffwrt(3) diffwrt1(3)

disjoint(1)              listro.pl     disjoint(1)

doexpr(7)                iorout.pl     prlog(5)

dome(0)                  goals

domeineq(1)              goals

dorun(2)                 invoca.pl     foreach(5)

dottoand(2)              press         maximum(2) dottoand(2)

easysol(3)               press         sol11(3)

enter(6)                 struct.pl     prcomplex(6) enter(6)

err(2)                   iorout.pl     error(3)

error(3)                 iorout.pl

errormess(2)             undefined     err(2)

eval(2)                  eval          sineval(2)    taneval(2)     arcsineval(2
                                       arccoseval(2) odd(1) even(1)
```

evall(1)	undefined	evall(2)
evall(2)	eval	fl3(2) eval(2) evall(2)
even(1)	eval	
expeqn(1)	goals	
findall(3)	invoca.pl	
findbnd(3)	press	
findmax(3)	press	findbnd(3)
fixvar(2)	press	solveineq(3) solvel1(3)
fl1(2)	simp	flush(2) fl1(2)
fl2(2)	simp	flush(2) fl2(2)
fl3(2)	simp	flush(2) fl3(2)
flush(2)	simp	bag_routines(2)
for(2)	invoca.pl	action(3) for(2)
forall(2)	invoca.pl	
foreach(5)	invoca.pl	
freeof(2)	simp	sol11(3) isolate(3) lin(4) quad(5) changeunknown(4) diffwrt1(3)
freeofunks(1)	simp	
gcc(1)	miscel.pl	simplify1(2) solvel1(3) simsolvel(3)
gensym(2)	miscel.pl	cgensym(2) arbint(1) identifier(1) diffwrt1(3)
getcode(3)	iorout.pl	writefs(2)
getdigits(4)	iorout.pl	getcode(3) getdigits(4)
getop(2)	invoca.pl	acollect(2)
given(1)	goals	s_or_g(1)
givesneg(3)	press	
identifier(1)	simp	changeunknown(4)
incline(3)	goals	classify(2)
intermediate(1)	goals	unknown(1)

```
intermediates_in(2)     press         findbnd(3)

intersect(3)            setrou.pl     intersect(3)

intexp(3)               eval          eval1(2) intexp(3)

isolate(3)              press         easysol(3) isolate(3)

isolax(4)               press         isolate(3)

isolax1(4)              press         isolax(4)

known(1)                known         tsimpax(2)

last(2)                 listro.pl     last(2)

lcollect(2)             invoca.pl     findall(3) lcollect(2) acollect(2)

least_dom(2)            press         collect(3) attract(3)

lin(4)                  press         easysol(3) lin(4) quad(5)

listtoset(2)            listro.pl     listtoset(2) remove_dups(2)

loseqn(1)               goals

loop(0)                 goals

loopineq(1)             goals

mapand(3)               applic.pl     mapand(3) solveineq(3)

maplist(3)              applic.pl     maplist(3) recurse(3)  eval(2)  findmax(3
                                      match(2) range(3)

match(2)                press         fl2(2) cancelling(2) isolate(3) attract(3
                                      match(2) changebas(4)

maximum(2)              press         solveineq(3)

maximum1(2)             press         maximum(2) maximum1(2)

measure(2)              goals         known(1) classify(2)

member(2)               setrou.pl     listtoset(2)    intersect(3)     member(2
                                      subtract(3)  union(3)  flush(2) subterm(2
                                      unknowns(1)

memberchk(2)            setrou.pl     disjoint(1) memberchk(2) subset(2) bind(3

mess(0)                 main2.pl      version(0)

min(3)                  press         loop(0) dome(0)

mlapply(2)              multil.pl     mlmaplist(2) mlmaplist(3) mlmaplist(4)
```

mlmaplist(2)	multil.pl	mlmaplist(2)
mlmaplist(3)	multil.pl	mlmaplist(3)
mlmaplist(4)	multil.pl	mlmaplist(4)
mlmember(2)	multil.pl	mlmember(2)
mlselect(3)	multil.pl	mlselect(3)
natnum(1)	eval	fl3(2) odd(1) even(1)
negate(2)	eval	eval1(2)
negative(1)	simp	tsimpax(2) arcsineval(2) arccoseval(2)
nextto(3)	listro.pl	nextto(3)
nl4(0)	goals	
nlc(1)	basini	
nobt(1)	invoca.pl	binding(2) lcollect(2) acollect(2) acollect2(3) gcc(1)
nobtwritefs(2)	iorout.pl	writef(2)
non_neg(1)	simp	arcsineval(2) arccoseval(2)
non_reflex(1)	simp	
nonzero(1)	simp	ntsimpax(2)
normalize(2)	simp	solveineq(3) solve11(3)
normax(2)	simp	
ntsimpax(2)	simp	simplify2(2) simplify3(2)
numlist(3)	listro.pl	numlist(3)
obtuse(1)	simp	
occ(3)	struct.pl	freeof(2) singleocc(2) contains(2 collect(3) attract(3) changeunknown(4)
occ2(3)	struct.pl	occ(3) occ2(3)
odd(1)	eval	intexp(3)
onexars(3)	press	diffwrt1(3)
pairint(4)	simp	fl3(2)
partition(2)	goals	range(3)

perm(2)	listro.pl	perm(2) splitperm(3)
perm2(4)	listro.pl	cancelling(2) reciprocal(2) fl3(2) lin(4) quad(5) match(2)
positive(1)	simp	negative(1) tsimpax(2) tidyax(2)
postulate(1)	miscel.pl	
prcomplex(6)	struct.pl	prdep(6)
prconj(1)	iorout.pl	prconj(1) action(3)
prdep(6)	struct.pl	prtodepth(3) prdep2(6)
prdep2(6)	struct.pl	prcomplex(6) prdep2(6)
prels(2)	iorout.pl	prexpr(1) prels(2)
prexpr(1)	iorout.pl	action(3)
prgen(2)	struct.pl	enter(6)
prlist(1)	iorout.pl	prlist(1) action(3)
prlog(5)	iorout.pl	prexpr(1) doexpr(7)
prove(1)	press	givesneg(3) bigger(2) simp3(0)
prsimple(5)	struct.pl	prdep(6)
prtodepth(3)	struct.pl	
pulltab(0)	goals	
quad(4)	known	range(3)
quad(5)	press	specialcase(3) quad(5)
quadeqn(1)	goals	
quantity(1)	goals	known(1)
range(3)	known	classify(2)
reciprocal(2)	simp	fl2(2)
recomp(2)	simp	recomp(2) bas_routines(2) collect(3) attract(3) match(2) changebas(4)
recurse(3)	simp	simplify2(2) tidy(2) normalize(2)
rem_simp(2)	undefined	simplify1(2)
remove_dups(2)	listro.pl	onexars(3)

repeat(1)	invoca.pl	
retract(2)	miscel.pl	
rev(2)	listro.pl	
revconc(3)	listro.pl	rev(2) revconc(3)
rewrite(3)	simp	tidy(2) normalize(2)
run(0)	goals	
s_or_s(1)	press	
select(3)	listro.pl	perm(2) select(3) fl1(2) pairint(4) twofrom(4) match(2)
seteq(2)	setrou.pl	
simp3(0)	goals	
simpeqns(0)	goals	
simplify(2)	simp	tsimpax(2) arctaneval(2) prove(1) sol11(3)
simplify1(2)	simp	positive(1) non_neg(1) nonzero(1) acute(1) obtuse(1) non_reflex(1) simplify(2)
simplify2(2)	simp	simplify2(2)
simplify3(2)	simp	simplify2(2) simplify3(2)
simpfull(0)	goals	
simsolve(2)	press	simpeqns(0) simpfull(0) fulltab(0) nl4(0) car(0) tower1(0) train(0) tower2(0)
simsolve1(3)	press	simsolve(2) simsolve1(3)
sineval(2)	eval	eval1(2) sineval(2) coseval(2) taneval(2)
sineval1(2)	eval	sineval(2) arcsineval(2) arccoseval(2)
singleocc(2)	simp	isolate(3)
slope(2)	goals	range(3)
smaller(2)	press	
snorm(2)	known	range(3)
sol11(3)	press	sol11(3)
solve11(3)	press	findbnd(3) prove(1) sol11(3) findmax(3) simsolve1(3) logeqn(1) expeqn(1)

		trigeqn(1) coueqn(1) coseqn(1) sqrteqn(1)
		quadeqn(1)

solveineq(3)	press	min(3) stvineq(1) blocineq(1) domeineq(1)
		conjineq(1) loopineq(1) bloc(0)

some(2)	press	
some(2)	applic.pl	some(2) maximum1(2) some(2)
sought(1)	goals	unknown(1) s_or_g(1)
special(2)	iorout.pl	writefs(2)
specialcase(3)	press	sol11(3)
splitperm(3)	simp	match(2)
sqrteqn(1)	goals	
sqrteval(2)	eval	eval1(2)
stats(1)	goals	
stvineq(1)	goals	
subgoal(2)	basini	
sublist(3)	applic.pl	sublist(3) intermediates_in(2) findmax(3 least_dom(2) onexargs(3)
subset(2)	setrou.pl	seteq(2) subset(2)
subst(3)	struct.pl	subst_mess(3) subst(3) subst2(4 changeunknown(4) diffwrt1(3)
subst2(4)	press	
subst_mess(3)	struct.pl	sol11(3) givesneg(3) simsolve1(3)
subterm(2)	simp	subterm(2) changeunknown(4)
subtract(3)	setrou.pl	subtract(3)
succ(2)	miscel.pl	
taketop(3)	multil.pl	mlmaplist(2) mlmaplist(3) mlmaplist(4 mlmember(2) mlselect(3) taketop(3)
taneval(2)	eval	eval1(2)
thnot(1)	invoca.pl	forall(2)
tidy(2)	simp	simplify1(2) solveineq(3) sol11(3 easysol(3) subst2(4) isolate(3) collect(3

```
                                      attract(3) specialcase(3)  lin(4)  quad(5)
                                      diffwrt(3) simsolve(2)

tidyax(2)               simp

tlim(1)                 iorout.pl

tower1(0)               goals

tower2(0)               goals

trace(2)                iorout.pl     changeunknown(4)

trace(3)                iorout.pl     trace(2)     subst_mess(3)     simplify(2)
                                      arbint(1)  min(3)  solveineq(3) findbnd(3)
                                      solve11(3) sol11(3) easysol(3)  isolate(3)
                                      collect(3)    attract(3)    specialcase(3)
                                      changeunknown(4)  diffwrt(3)   simsolve(2)
                                      simsolve1(3) known(1) stats(1)

train(0)                goals

trdep(2)                iorout.pl

trigeqn(1)              goals

tsimpax(2)              simp          simplify2(2) simplify3(2)

twofrom(4)              simp          f12(2) f13(2) changebag(4)

union(3)                setrou.pl     union(3) wordsin(2)

unionlist(2)            known         range(3) unionlist(2)

unit(1)                 miscel.pl

unknown(1)              simp          freeofunks(1)

unknowns(1)             press         simsolve(2)

valof(2)                assign.pl     trace(3) trdep(2) cvalof(3)

valofdd(2)              undefined     valof(2)

variables(2)            struct.pl

variables(3)            struct.pl     variables(2) variables(3)

vchk(3)                 struct.pl     variables(3) vchk(3)

version(0)              main2.pl

wordsin(2)              eval          wordsin(2) intermediates_in(2)

writef(1)               iorout.pl
```

writef(2)	iorout.pl	error(3) err(2) trace(3) writef(1)
writefs(2)	iorout.pl	nobtwritefs(2) writefs(2)
xattrax(4)	press	attract(3)

```
From Richard[400,422] on December 19, 1979 at 5:48 PM
My files start with this:
/* FILNAM version 1 of dd/mm/yy

   comments (not yet very elaborate, but I'll copy Lawrence)
*/
- two blank lines -

My other comments arein the following forms:

/* short title */
f(......) :-
        ..... /* inline */
        ... . /* inline */

/* longer comments for medium-scale
   sections of program occupy several
   lines.  They are always indented
   so that I can see the brackets.
*/
/* I never put anything after '*/' on the
   same line, for that reason.
*/
```

Comments for sections of code, particularly for procedures, immediately precede the section they describe. One blank line may occasionally be inserted for clarity. Inline comments always immediately follow the term they clarify.

There is one glaring exception to these rules:

```
p(X, Y, Z) :-
/*db:   write(...), write(...),
*/      -normal code-
        -normal code- .
```

However, debugging code thus commented out will never appear (I hope) in programs distributed by me. The other debugging technique I use is

```
        (debug -> ..., ..., ...; true),
```

which can be turned on by "assert(debug)" and off by "retract(debug)".

S

Descrition of utility procedures.

Lawrence Byrd

FILE util.

This file describes the routines in the new utilities file UTIL giving a brief description of the function of each, and also indicating how they differ from any older versions in USVW.PL.
The following conventions are employed :

```
  A1,A2 ...An               :  Represent the arguments to the routine
                               in question.
  AS OLD                    :  The routine in question is identical to
                               the old version in USVW.PL.
  NEW                       :  The routine in question is either completely
                               new or is radically different from any old
                               version of the same name.
  DIFF but CP identical     :  The routine in question performs the same
                               function as the old version and has the
                               same Calling Protocol , but it has been
                               rewritten in some way.
  DIFF and CP changed       :  The routine in question performs roughly
                               the same function as the old version but
                               the Calling Protocol has been changed
                               (This may include,for example, the number
                               and types of the arguments.)
  DELETED                   :  The routine in question is not to be
                               found in UTIL.
```

BRICK io routines.

```
tlim            The level for tracing is set to A1.
                DIFF bu CP identical.

trace           Two versions :
                3 args  Print out a writef message using A1 and A2
                        if the current trace level is greater than or
                        equal to A3.
                2 args  Print out a writef message using A1 and []
                        if the current trace level is greater than or
                        equal to A2.
                DIFF and CP changed.

trdep           Call A1 if the current trace level is greater than or equal
                to A2. (i.e. Perform a trace level dependent action).
                NEW.

prconj          Print A1 (a conjunction), a conjunct per line. (prconj is
                available from within writef and therefore trace as well).
                (Note: & is the functor for conjunctions).
                NEW.

prlist          Print A1 (a list), an item per line. (prlist is available
```

from within writef and therefore trace as well).
NEW.

writef — Print onto the current output according to the format string A1 (quoted atom) and the argument list A2. See the file writef.doc[400,441,doc] for details.
NEW.

writefs — Similar to writef except that the format string is a proper string (list) rather than a quoted atom. Provides basis for writef.
NEW.

BRICK list routines.

append — A3 is the list formed by appending A1 and A2.
AS OLD.

disjoint — A1 (a list) is pairwise disjoint.
AS OLD.

last — A1 is the last element of the list A2.
AS OLD.

listtoset — A2 is the set of elements of the list A1. i.e. A2 is a list with no duplicates.
DIFF but CP identical.

nextto — A1 and A2 are next to eachother in the list A3.
AS OLD.

numlist — A3 is a list of successive integers from A1 to A2.
NEW.

perm — Bactracking will vary A2 over all the possible permutations of the elements of the list A1. (i.e. A2 will be set equivalent to A1).
NEW (Bundy).

perm2 — Similar to perm except for only two elements (A1 & A2 -> A3 & A4).
NEW (Bundy).

remove_dups — Equivalent to listtoset. A2 is the list A1 with all the duplicates removed.
DIFF but CP identical.

rev — A2 is the list A1 with the elements in reverse order.
AS OLD.

select — A1 is a member of the list A2, A3 is the list A2 with the element A1 removed.
NEW (Bundy).

BRICK set routines.

intersect
A3 (a set) is the set-intersection of the sets A1 and A2.
AS OLD.

member
A1 is a member of the list/set A2.
NEW.

memberchk
A1 is a member of the list/set A2. (But unlike member, memberchk does not allow backtracking).
NEW.

seteq
The sets A1 and A2 are equivalent.
AS OLD.

subset
A1 (a set) is a subset (a la <=) of the set A2.
AS OLD.

subtract
A3 (a set) is equivalent to the set formed by subtracting the sets A1 and A2 (A1-A2=A3).
AS OLD.

union
A3 (a set) is the set-union of the sets A1 and A2.
AS OLD.

BRICK invocation routines.

&
A1 AND A2. Goal conjunction. & is an operator.
NEW.

\\
A1 OR A2. Exclusive disjunction of goals. \\ is an operator.
NEW.

any
Call the members of A1 in sequence until any one of them succeeds.
AS OLD.

binding
return the A1'th binding of A2 (or fail). binding is the generalisation containing the old 'second'.
NEW.

findall
A3 is the list of all A1's such that A2. (findall will work recursively and any number of the arguments of A2 can be collected via A1).
DIFF and CP changed.

for
Call A2 A1 times.
NEW.

forall
(X)(A1(X)->A2(X)). For all X A1 implies A2.
DIFF but CP identical.

foreach
For each A1 call A2 and collect together the A4's (using the functor/operator A3) to form the term A5. foreach is

a more general version of sumeach.(foreach allows recursion and is not restricted to merely summing).
NEW.

nlc
Non loop call. Call A1 unless the current goal is a subgoal of A1. (Note: subgoal_of rather than one of the 'subgoal' alternatives (see later)).
AS OLD.

nobt
No backtracking. Call A1 but cut any backtracking.
NEW.

repeat
Repeatedly call A1 in a failure driven (backtracking) loop until it fails.

BRICK application routines.

apply
Apply the (possibly partly applied) predicate A1 to the arguments A2 (a list). The convention is that the extra arguments (from the list) go AFTER any arguments already in A1. (This is the new convention).
DIFF and CP changed.

checkand
Apply the predicate A1 to every element of the conjunction A2 in turn. (see apply).
NEW (Bundy).

checklist
Apply the predicate A1 to every element of the list A2 in turn. (see apply).
AS OLD.

mapand
Apply the predicate A1 to the corresponding elements of the two conjunctions A2 and A3 ,in turn. (see apply).
NEW (Bundy).

maplist
Apply the predicate A1 to the corresponding elements of the two lists A2 and A3 ,in turn. (see apply).
AS OLD.

convlist
Apply the predicate A1 to each element of the list A2 in turn, and when A1 succeeds place the second applied argument in A3. (x)[(Ey)(y{A2 & A1(y,x) .-> x{A3]
AS OLD.

sublist
Apply the predicate A1 to each element of the list A2 in turn, and when A1 succeeds place that element in A3 (a list). A3 is therefore the sublist of all elements of A2 having the property A1. (x)[x{A2 & A1(x) .-> x{A3]
AS OLD.

some
Apply the predicate A1 to succesive elements of the list A2 until A1 succeeds (or fail). (Ex)[x{A2 & A1(x)]
AS OLD.

BRICK multilist routines.

mlapply
Apply the (possibly partly applied) predicate A1 to the arguments A2 (a list). Extra arguments (from A2) are placed AFTER any arguments already in A1.
(apply now also uses this convention so that apply and mlapply are now identical).
NEW.

mlmaplist
The predicate A1 is ml-applied (see mlapply) to the 'lines' of the 'table' A2 (a list of lists; see mlmember), sequentially a line at a time. There are several versions of mlmaplist allowing certain methods of intercommunication between different applications as the lists are traversed.
NEW.

mlmember
A1 is a list which is a 'member line' of the list of lists A2. A2 can be considered as a table with 'lines' which are lists of corresponding elements in the lists of A2. mlmember traverses all the lists of A2 simultaneously.
NEW.

mlselect
A1 is a list which is a 'member line' of the list of lists A2 (see mlmember). A3 is a list of lists equivalent to A2 except that the line A1 is missing.
NEW.

BRICK assignment routines.

The following routines provide facilities for pseudo-assignment; they are of course very bad things to use in your pure and spotless Prolog programs - but they enable you to hide odd flags etc. The bindings are just assertions in the data-base so they are none too efficient and the identity of variables are lost. (All variables are replaced by new and different variables when the binding is asserted).

becomes
A binding is created between A1 and A2. (i.e. The "variable" A1 has the term A2 assigned to it).
NEW.

valof
The value of A1 is A2. (Recovers the last binding created by becomes),

cvalof
The value of A1 is A3, unless A1 has not been given a value in which case A2 is unified with A3.
NEW.

BRICK error handler.

error

An error message for error type A1 is printed, using A2 as the (writef) List. A3 is then called. (If no errormess is known for this type of error then a default printout is performed).
NEW.

BRICK miscellaneous routines.

casserta

A1 is asserta'd unless it is already true.
AS OLD.

cassertz

A! is assertz'd unless it is already true.
AS OLD.

clean

The log file (prolog.log) is emptied.
NEW.

concat

A3 (an atom) is equal to the atoms A1 and A2 concatenated together. (similar to append except for atoms rather than lists - A1 and A2 must be instantiated).
AS OLD.

continue

Equivalent to true, except slightly more mnemonic in certain circumstances.
NEW.

cretract

If A1 is true then retract it.
NEW.

\=

A1 and A2 do not unify (identical to diff). \= is an operator.
NEW.

diff

A1 and A2 are different (non-unifiable).
AS OLD.

gcc

Garbage collect call. A1 will be called and even if it succeeds all storage in the local and global stacks used by A1 will be recovered. Any backtracking possibilities will be cut.
DIFF but CP identical.

gensym

A2 should be uninstantiated - it becomes instantiated to an atom which is constructed from the atom A1 with a new integer postfix. (e.g. gensym(block,B) would instantiate B to block1, then gensym(block,BB) would instantiate BB to block2).
NEW.

cgensym

Only gensym (using A1) if A2 is not already instantiated.
NEW.

postulate

A1 is asserta'd,but it will be retract'ed on backtracking. (Beware - of cutting out the backtracing through postulate thus leaving A1 in the data-base).
AS OLD.

retract — (two args) The clause A1 :- A2 is retracted. (A2 = true is equivalent to a unit clause). This version of retract allows A2 to match against arbitrary clause bodies unlike the system version.
NEW.

(Only 'exact' around at the moment)

subgoal — There are four versions of subgoal all of which perform some sort of check to see if the current goal has a certain ancestor. By 'current goal' is meant the level of the call to 'subgoal' in the particular clause.

- subgoal(exact,A2) - The current goal is a susgoal of a parent which exactly matches A2.
- subgoal(ignore,A2,A3) - Ignoring A2 ancestors the current goal is a subgoal of A3.(i.e. if A2 = 1 then the head of the clause containing 'subgoal' will be ignored).
- subgoal(ignore,A2) - Equivalent to subgoal(ignore,1,A2).
- subgoal(skip,A2,A3) - Look to see if A3 is an ancestor but skip the first A2 matches. (i.e. there are more than A2 ancestors which will unify with A3.)

NEW.

succ — A2 is the (integer) successor of A1. (One or other must be instantiated).
NEW.

unit — A1 is a unit clause.

variables — A2 is a list which is the SET (no repetitions) of all the variables in the term A1.
NEW.

List of old routines not mentioned above.

pr	DELETED. (use writef).
ppr	DELETED. (use writef).
nwl	DELETED. (use trace or trdep).
seltrace	DELETED. (use debug package).
groundtest	Moved to BRICK search control. (LOGIC).
functest	Moved to BRICK search control. (LOGIC).
silly	Moved to BRICK search control. (LOGIC).
seperate	DELETED. (use multilist techniques).
sortout	DELETED.
split	DELETED.

isok	Moved to BRICK search control. (LOGIC).
member1	DELETED.
find	DELETED. (use multilist techniques).
members	DELETED.
bound	DELETED. (use Evaluable - nonvar).
nonvar	DELETED. (use Evaluable - nonvar).
word	DELETED. (use Evaluable - atom).
cond	DELETED. (use Evaluable - ->).
gensym	Moved to BRICK normal form. (LOGIC).
current	DELETED.
cgensym	Moved to BRICK normal form. (LOGIC).
reccl	DELETED.
findcl	DELETED.
onceandfail	DELETED.
sumeach	DELETED. (use foreach).
sum1	DELETED.
second	DELETED.
twice	DELETED.

Lawrence Byrd
[400,441]
Dept AI
Edinburgh

SIMPLE INTRODUCTION TO WRITEF

writef is a formatted write utility available when running UTIL[400,444]
The following documentation is both minimal and interim.

There are two versions - one with one argument and the other with two.
These are as follow :

```
writef(Format) :- writef(Format,[]).

writef(Format,Item_list) :- ...............
```

Format is an atom which is interpreted as a string of characters, (since it will contain all sorts of characters it will normally have to be quoted). Item_list is a list of any old Prolog terms. writef turns the Format atom into a list of characters and then runs down this list outputting the characters onto the current output device. Certain character sequences have a special meaning however :

SPECIAL CHARACTERS

All the following sequences result in a particular (rather difficult to use) character being output.

```
\n      -       newline (carriage return, linefeed).
\l      -       linefeed.
\r      -       carriage return.
\t      -       tab.
\\      -       the character "\"
\%      -       the character "%"
\xxx    -       where xxx is a decimal integer (i.e. between
                1 and 3 digits - no more than 3 are ever eaten up
                The character with ASCII code xxx (decimal!!)
                is output.
```

SPECIAL FORMAT ITEMS

All the following take items off the Item_list and output them in a particular way.

```
%t      -       The next item is written (using write -
                mnemonic is "term").
%l      -       The next item is a list which is written
                one element to a line with an indent of 4.
%c      -       The next item is a conjunction (using & as
                a functor) which is written one element to
                a line with an indent of 4.
%e      -       The next item is a logical expression built
                up with functors & (AND) and # (OR), It is
                written in a nice understandable format.
%n      -       The next item is an integer, The ASCII character
                with that (decimal) code is written.
```

%r — Two items are taken, The first is written (using write) N times where N is the second item (an integer).

%f — A ttyflush is performed (No items are used).

EXAMPLES

The following examples show how writef is used :

```
  writef('Hello there!!!! \n\n').
  writef('The conjunction %c flattens to the list %l',[Conj,List]).
  writef('The answer for :\t%t \nis:%e',[Question,Answer]).
  writef('What do you think ? %f').
  writef('TITLE\n%r\n And now....\7 \7\n',[-,5]).
```

If you are unsure as what some of these do - try them out!! (Remember to instantiate the variables (e.g. Conj), to som

From Lawrence[400,441] on November 16, 1979 at 6:44 PM

Hello,

Just a quick note about undocumented things in UTIL :

variables(Term,Vset)

When given any Prolog term this returns the set (ie no duplications) of Prolog variables contained in the Term.

subst(Old=New,Oldterm,Newterm)

This is the routine used in PRESS, it applies the substitution Old=New to Oldterm to produce Newterm. It is quite general and can handle Prolog variables OK (ie it doesn't get confused and allow unification to occur - so for example the substitution X=Y when X and Y are both (uninstantiated) variables does the expected thing of replacing all occurances of the variable X by Y (but not any old thing which unifies with X - ie everythi

These may be useful for your QA program. (indeed I see you have writte your own).

Lawrence

(ps : [-foo] is a short way of doing reconsult(foo))