### PROLOG CROSS REFERENCE LISTING

The PRESS system

<table>
<thead>
<tr>
<th>PREDICATE</th>
<th>FILE</th>
<th>CALLED BY</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;(2)</code></td>
<td><code>invoca.pl</code></td>
<td><code>fixvar(2)</code></td>
</tr>
<tr>
<td><code>&lt;=(2)</code></td>
<td><code>miscel.pl</code></td>
<td></td>
</tr>
<tr>
<td><code>\\(2)</code></td>
<td><code>invoca.pl</code></td>
<td></td>
</tr>
<tr>
<td><code>acollect(2)</code></td>
<td><code>invoca.pl</code></td>
<td><code>foreach(5)</code></td>
</tr>
<tr>
<td><code>acollect2(3)</code></td>
<td><code>invoca.pl</code></td>
<td><code>acollect(2) acollect2(3)</code></td>
</tr>
<tr>
<td><code>action(3)</code></td>
<td><code>iorout.pl</code></td>
<td><code>writefs(2)</code></td>
</tr>
<tr>
<td><code>acute(1)</code></td>
<td><code>simp</code></td>
<td></td>
</tr>
<tr>
<td><code>allempy(1)</code></td>
<td><code>multil.pl</code></td>
<td><code>mlmaplist(2) mlmaplist(3) mlmaplist(4)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>allempy(1)</code></td>
</tr>
<tr>
<td><code>andtodot(2)</code></td>
<td><code>press</code></td>
<td><code>maximum(2) andtodot(2)</code></td>
</tr>
<tr>
<td><code>angle(3)</code></td>
<td><code>goals</code></td>
<td><code>classify(2)</code></td>
</tr>
<tr>
<td><code>any(1)</code></td>
<td><code>invoca.pl</code></td>
<td><code>any(1)</code></td>
</tr>
<tr>
<td><code>any1(3)</code></td>
<td><code>simp</code></td>
<td><code>any1(3) collect(3) attract(3)</code></td>
</tr>
<tr>
<td><code>append(3)</code></td>
<td><code>listro.pl</code></td>
<td><code>append(3) apply(2) mlapply(2) concat(3)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>splitperm(3) collect_ans(3)</code></td>
</tr>
<tr>
<td><code>apply(2)</code></td>
<td><code>applic.pl</code></td>
<td><code>checkand(2) checklist(2) mapand(3)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>maplist(3) convlist(3) some(2) sublist(3)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>any1(3) rewrite(3) some(2)</code></td>
</tr>
<tr>
<td><code>arbint(1)</code></td>
<td><code>simp</code></td>
<td></td>
</tr>
<tr>
<td><code>arccoseval(2)</code></td>
<td><code>eval</code></td>
<td><code>eval1(2)</code></td>
</tr>
<tr>
<td><code>arcsineval(2)</code></td>
<td><code>eval</code></td>
<td><code>eval1(2) arctaneval(2)</code></td>
</tr>
<tr>
<td><code>arctaneval(2)</code></td>
<td><code>eval</code></td>
<td><code>eval1(2)</code></td>
</tr>
<tr>
<td><code>attract(3)</code></td>
<td><code>press</code></td>
<td><code>sol11(3)</code></td>
</tr>
<tr>
<td>Function</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>attract(4)</td>
<td>Press</td>
<td>attract(3)</td>
</tr>
<tr>
<td>bas_routines(2)</td>
<td>simp</td>
<td>simplify2(2) tidy(2)</td>
</tr>
<tr>
<td>becomes(2)</td>
<td>assign.pl</td>
<td>tlim(1) sensym(2)</td>
</tr>
<tr>
<td>bigger(2)</td>
<td>Press</td>
<td>smaller(2)</td>
</tr>
<tr>
<td>bind(3)</td>
<td>struct.pl</td>
<td>bind(3)</td>
</tr>
<tr>
<td>bindings(2)</td>
<td>invoca.pl</td>
<td></td>
</tr>
<tr>
<td>bloc(0)</td>
<td>soals</td>
<td></td>
</tr>
<tr>
<td>blocinea(1)</td>
<td>soals</td>
<td></td>
</tr>
<tr>
<td>cancelling(2)</td>
<td>simp</td>
<td>f12(2) reciprocal(2)</td>
</tr>
<tr>
<td>car(0)</td>
<td>soals</td>
<td></td>
</tr>
<tr>
<td>casserta(1)</td>
<td>miscel.pl</td>
<td>simplify1(2)</td>
</tr>
<tr>
<td>cassertz(1)</td>
<td>miscel.pl</td>
<td></td>
</tr>
<tr>
<td>csensym(2)</td>
<td>miscel.pl</td>
<td></td>
</tr>
<tr>
<td>changebas(4)</td>
<td>Press</td>
<td>collect(3) attract(3)</td>
</tr>
<tr>
<td>changeunknown(4)</td>
<td>Press</td>
<td>solii(3)</td>
</tr>
<tr>
<td>checkand(2)</td>
<td>applic.pl</td>
<td>checkand(2)</td>
</tr>
<tr>
<td>checklist(2)</td>
<td>applic.pl</td>
<td>checklist(2) f13(2) eval(2) maximum1(2) run(0)</td>
</tr>
<tr>
<td>classify(2)</td>
<td>known</td>
<td>known(1)</td>
</tr>
<tr>
<td>clean(0)</td>
<td>miscel.pl</td>
<td></td>
</tr>
<tr>
<td>cleanstack(0)</td>
<td>invoca.pl</td>
<td>dorun(2)</td>
</tr>
<tr>
<td>cnorm(2)</td>
<td>known</td>
<td>range(3)</td>
</tr>
<tr>
<td>collax(3)</td>
<td>Press</td>
<td>collect(3)</td>
</tr>
<tr>
<td>collect(3)</td>
<td>Press</td>
<td>solii(3)</td>
</tr>
<tr>
<td>collect_ans(3)</td>
<td>Press</td>
<td>findmax(3) collect_ans(3)</td>
</tr>
<tr>
<td>combine(3)</td>
<td>known</td>
<td>unionlist(2)</td>
</tr>
<tr>
<td>concat(3)</td>
<td>miscel.pl</td>
<td>sensym(2)</td>
</tr>
<tr>
<td>concavity(2)</td>
<td>soals</td>
<td>range(3)</td>
</tr>
</tbody>
</table>
The PRESS system

conjineq(1) soals
const(1) init wordsin(2)
contains(2) simp freeofunks(1) isolate(3) collect(3)
continue(0) miscel.pl
convlist(3) applic.pl convlist(3)
cosean(1) soals
coseval(2) eval eval1(2) taneval(2)
coueasn(1) soals
cretract(1) miscel.pl
cvalof(3) assign.pl sensym(2)
decomP(2) simp decomP(2) bas_routines(2) collect(3)
 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) soals
domeinea(1) soals
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
enormess(2) undefined err(2)
eval(2) eval sineval(2) taneval(2) arcsineval(2)
arccoseval(2) odd(1) even(1)
eval1(1) undefined eval1(2)
eval1(2) eval f13(2) eval(2) eval1(2)
even(1) eval
expen(1) soals
findall(3) invoca.pl
findbnd(3) press
findmax(3) press findbnd(3)
fixvar(2) press solveineq(3) solve11(3)
f11(2) simp flush(2) f11(2)
f12(2) simp flush(2) f12(2)
f13(2) simp flush(2) f13(2)
flush(2) simp bas_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
scc(1) miscel.pl simplify1(2) solve11(3) simsolve1(3)
sensym(2) miscel.pl cnsensym(2) arbint(1) identifier(1)
diffwrt1(3)
setcode(3) iorout.pl writes(2)
setdisits(4) iorout.pl setcode(3) setdisits(4)
setop(2) invoca.pl acollect(2)
siven(1) soals s_or_s(1)
sivesnes(3) press
identifier(1) simp changeunknown(4)
incline(3) soals classify(2)
intermediate(1) soals unknown(1)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intermediates_in</td>
<td>press</td>
</tr>
<tr>
<td>intersect</td>
<td>setrou.pl</td>
</tr>
<tr>
<td>interexp</td>
<td>eval</td>
</tr>
<tr>
<td>isolate</td>
<td>press, easyisol</td>
</tr>
<tr>
<td>isolax</td>
<td>press</td>
</tr>
<tr>
<td>isolax1</td>
<td>press</td>
</tr>
<tr>
<td>known</td>
<td>known</td>
</tr>
<tr>
<td>last</td>
<td>listro.pl</td>
</tr>
<tr>
<td>lcollect</td>
<td>invoca.pl</td>
</tr>
<tr>
<td>least_dom</td>
<td>press, collect</td>
</tr>
<tr>
<td>lin</td>
<td>press, easyisol, lin_quad</td>
</tr>
<tr>
<td>listtoset</td>
<td>listro.pl, listtoset</td>
</tr>
<tr>
<td>loosean</td>
<td>soals</td>
</tr>
<tr>
<td>loop</td>
<td>soals</td>
</tr>
<tr>
<td>loopineq</td>
<td>soals</td>
</tr>
<tr>
<td>mapand</td>
<td>applic.pl, mapand</td>
</tr>
<tr>
<td>maplist</td>
<td>applic.pl, maplist</td>
</tr>
<tr>
<td>match</td>
<td>press, fl2, cancelling, isolate,</td>
</tr>
<tr>
<td></td>
<td>attract</td>
</tr>
<tr>
<td>maximum</td>
<td>press, solveineq</td>
</tr>
<tr>
<td>maximum1</td>
<td>press, maximum, maximum1</td>
</tr>
<tr>
<td>measure</td>
<td>soals, known</td>
</tr>
<tr>
<td>member</td>
<td>setrou.pl, listtoset, intersect,</td>
</tr>
<tr>
<td></td>
<td>member</td>
</tr>
<tr>
<td></td>
<td>memberchk</td>
</tr>
<tr>
<td>mess</td>
<td>main2.pl</td>
</tr>
<tr>
<td>min</td>
<td>press, loop, dome</td>
</tr>
<tr>
<td>mapply</td>
<td>multil.pl, mlmaplist, mlmaplist,</td>
</tr>
<tr>
<td></td>
<td>mlmaplist</td>
</tr>
</tbody>
</table>
The PRESS system

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(l) eval f13(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)
n14(0) soals
n1c(1) basini
nobt(l) invoca, pl bindings(2) lcollect(2) scollect(2)
                 scollect2(3) sc(1)
nobtwrites(2) iorout, pl writef(2)
non_neg(1) simp arcsineval(2) arccoseval(2)
non_reflex(l) simp
nonzero(l) simp ntsimpax(2)
normalize(2) simp solveineq(3) solvel11(3)
normax(2) simp
ntsimpax(2) simp simplify2(2) simplify3(2)
numlist(3) listro, pl numlist(3)
obtuse(l) simp
occ(3) struct, pl ffreeof(2) singleocc(2) contains(2) collect(3) attract(3) changeunknown(4)
occ2(3) struct, pl occ(3) occ2(3)
odd(l) eval intexp(3)
onexars(3) press diffwrt1(3)
pairint(4) simp f13(2)
partition(2) soals range(3)
<table>
<thead>
<tr>
<th>perm(2)</th>
<th>listro.pl</th>
<th>perm(2) splitPerm(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>perm2(4)</td>
<td>listro.pl</td>
<td>cancellins(2) reciprocal(2) f13(2) lin(4) quad(5) match(2)</td>
</tr>
<tr>
<td>Positive(1)</td>
<td>simp</td>
<td>negative(1) tsimpax(2) tidyax(2)</td>
</tr>
<tr>
<td>Postulate(1)</td>
<td>miscel.pl</td>
<td></td>
</tr>
<tr>
<td>Pcomplex(6)</td>
<td>struct.pl</td>
<td>Prdep(6)</td>
</tr>
<tr>
<td>PrconJ(1)</td>
<td>iorout.pl</td>
<td>PrconJ(1) action(3)</td>
</tr>
<tr>
<td>Prdep(6)</td>
<td>struct.pl</td>
<td>Prtodepth(3) Prdep2(6)</td>
</tr>
<tr>
<td>Prdep2(6)</td>
<td>struct.pl</td>
<td>Pcomplex(6) Prdep2(6)</td>
</tr>
<tr>
<td>Prls(2)</td>
<td>iorout.pl</td>
<td>Prexpr(1) Prls(2)</td>
</tr>
<tr>
<td>Prexpr(1)</td>
<td>iorout.pl</td>
<td>action(3)</td>
</tr>
<tr>
<td>Prgen(2)</td>
<td>struct.pl</td>
<td>enter(6)</td>
</tr>
<tr>
<td>Prlist(1)</td>
<td>iorout.pl</td>
<td>Prlist(1) action(3)</td>
</tr>
<tr>
<td>Prlos(5)</td>
<td>iorout.pl</td>
<td>Prexpr(1) doexpr(7)</td>
</tr>
<tr>
<td>Prove(1)</td>
<td>Press</td>
<td>sivesnes(3) bisser(2) simp3(0)</td>
</tr>
<tr>
<td>Prsimple(5)</td>
<td>struct.pl</td>
<td>Prdep(6)</td>
</tr>
<tr>
<td>Prtodepth(3)</td>
<td>struct.pl</td>
<td></td>
</tr>
<tr>
<td>Pulltab(0)</td>
<td>soals</td>
<td></td>
</tr>
<tr>
<td>quad(4)</td>
<td>known</td>
<td>range(3)</td>
</tr>
<tr>
<td>quad(5)</td>
<td>Press</td>
<td>specialcase(3) quad(5)</td>
</tr>
<tr>
<td>quadean(1)</td>
<td>soals</td>
<td></td>
</tr>
<tr>
<td>quantity(1)</td>
<td>soals</td>
<td>known(1)</td>
</tr>
<tr>
<td>range(3)</td>
<td>known</td>
<td>classify(2)</td>
</tr>
<tr>
<td>reciprocal(2)</td>
<td>simp</td>
<td>f12(2)</td>
</tr>
<tr>
<td>reciprocal(2)</td>
<td>simp</td>
<td>recipro(2) bas_routines(2) collect(3) attract(3) match(2) changebas(4)</td>
</tr>
<tr>
<td>recurse(3)</td>
<td>simp</td>
<td>simplify2(2) tidy(2) normalize(2)</td>
</tr>
<tr>
<td>rem_simp(2)</td>
<td>undefined</td>
<td>simplify1(2)</td>
</tr>
<tr>
<td>remove_dups(2)</td>
<td>listro.pl</td>
<td>onexars(3)</td>
</tr>
</tbody>
</table>
The PRESS system

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) soals
s_or_s(1) Press
select(3) listro.pl perm(2) select(3) fl1(2) pairint(4)
twofrom(4) match(2)
setea(2) setrou.pl
simp3(0) soals
simpvars(0) soals
simplify(2) simp
simp
simplify1(2) simp
tsimpax(2) arctaneval(2) prove(1) sol11(3)
simplify2(2) simp
positive(1) non_neg(1) nonzero(1) acute(1)
obtuse(1) non_reflex(1) simplify(2)
simplify3(2) simp
simplify2(2) simplify3(2)
simpfull(0) soals
simsolve(2) Press simpeans(0) simpfull(0) pulltab(0) n14(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) soals 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) losean(1) expfan(1)
The PRESS system

troleaan(1) coveean(1) cosean(1) sarteaan(1)
quadean(1)
solveinea(3) Press
min(3) stvinea(1) blocinea(1) domeinea(1)
conbinea(1) loopinea(1) bloc(0)
some(2) Press
some(2)
applic,pl
some(2) maximum1(2) some(2)
sough(1)
soals
unknown(1) s_or_s(1)
special(2)
iorout,pl
writefs(2)
specialcase(3)
Press
sol11(3)
splitperm(3)
simp
match(2)
sarteaan(1)
soals
sarteav(2)
eval
eval1(2)
stats(1)
soals
stvinea(1)
soals
subsoal(2)
basini
sublist(3)
applic,pl
sublist(3) intermediates_in(2) findmax(3):
leasdom(2) onexas(3)
subset(2)
setrou,pl
setea(2) subset(2)
subst(3)
struct,pl
subst_mess(3)
subst(3)
subst2(4):
chansetunknown(4) diffwrt1(3)
subst2(4)
Press
subst_mess(3)
struct,pl
sol11(3) sivesnes(3) simsove1(3)
subterm(2)
simp
subterm(2) chansetunknown(4)
subtract(3)
setrou,pl
subtract(3)
succ(2)
miscel,pl
taketoP(3)
multil,pl
mlmamlist(2) mlmaplist(3) mlmaplist(4):
mlmamember(2) mlmselect(3) taketoP(3)
taneval(2)
eval
eval1(2)
thnot(1)
invoca,pl
forall(2)
tidy(2)
simp
simplify1(2) solveinea(3) sol11(3)
essysol(3) subst2(4) isolate(3) collect(3)
The PRESS system

tidyax(2)       simp

tlim(1)         iorout.pl

tower1(0)       soals

tower2(0)       soals

trace(2)        iorout.pl

trace(3)        iorout.pl

train(0)        soals

trdeP(2)        iorout.pl

trisean(1)      soals

tsimPax(2)      simp

twofrom(4)      simp

union(3)        setrou.pl

unionlist(2)    known

unit(1)         miscel.pl

unknown(1)      simp

unknowns(1)     press

valof(2)        assign.pl

valofdd(2)      undefined

variables(2)    struct.pl

variables(3)    struct.pl

vchk(3)         struct.pl

version(0)      main2.pl

wordsin(2)      eval

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)
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 are in 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
   (debus -> ..., ..., ..., true),
   which can be turned on by "assert(debus)" and off by "retract(debus)".
Description of utility procedures.

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 Callins 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 Callins 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 but 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 A2 if the current trace level is greater than or equal to A2.

DIFF and CP changed.

trder

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 strings
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 strings 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 1s <=) 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 conjuction, & 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.

bindings 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)).
NEW.

nbt
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) so 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 successive 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 mllapply are now identical).

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

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.

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.

BRICK assignment routines.

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

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.

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 error message is known for this type of error then a default printout is performed).

NEW.

BRICK miscellaneous routines.

----------------------------------------
cassert  A1 is asserta'd unless it is already true.
        AS OLD.
cassertz A1 is assertz'd unless it is already true.
        AS OLD.
clean   The los file (prolos,los) 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.
scc     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.
sensym A2 should be uninstantiated - it becomes instantiated to an atom which is constructed from the atom A1 with a new integer postfix. (e.g. sensym(block,B) would instantiate B to block1, then sensym(block,BB) would instantiate BB to block2).
        NEW.
csensym Only sensym (using A1) if A2 is not already instantiated.
        NEW.
Postulate A1 is asserta'd, but it will be retracta'd on backtracks.
        (Beware - of cutins out the backtracins through Postulate thus leaving A1 in the data-base).
        AS OLD.
retract 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)

subsoal There are four versions of subsoal 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 'subsoal' in the particular clause.

subsoal(exact,A2) - The current goal is a subgoal of a parent which exactly matches A2.

subsoal(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 'subsoal' will be ignored).

subsoal(ignore,A2) - Equivalent to subsoal(1,A2).

subsoal(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).

prpr 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.
Moved to BRICK search control, (LOGIC).

DELETED.

DELETED. (use multilist techniques).

DELETED.

DELETED, (use Evaluable - nonvar).

DELETED, (use Evaluable - nonvar).

DELETED. (use Evaluable - atom).

DELETED. (use Evaluable - -> ).

Moved to BRICK normal form, (LOGIC).

DELETED.

Moved to BRICK normal form, (LOGIC).

DELETED.

DELETED.

DELETED, (use foreach).

DELETED.

DELETED.

DELETED.

DELETED.

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
outputing 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 (carrisse return, linefeed).
- \l linefeed.
- \r carrisse 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 \texttt{writef} is used:

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

Just a quick note about undocumented things in UTIL:

variables(Term,Vset)
When given any Prolog term this returns the set (ie no duplicates) 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 set 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 occurrences of the variable X by Y (but not any old things which unifies with X - ie everythi

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

Lawrence

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