```
~CA4.LSP: W. L. Johnson 10/79 Revision of Larry Birnbaum's CA4.MLI  8/79
~ Functions which are used in the actions of requests


~This is used in the actions of requests to spawn new requests.
~Request pool is tied to the con atom
(DF ACTIVATE (REQS)
  (LET (REQ (MAKE-REQUESTS REQS))
   (COND (!NEW-CON                    ~check if !NEW-CON has been set
          (PUTPROP !NEW-CON REQ 'ASSOC-REQS)))
   (ACTIVATE-POOL REQ)


~Intended for use mainly in the actions of special requests (those which
~test for some particular word), to alter the sense of some other word;
(DF ADD-TO-WORD-SENSE (REQS)
  (SETQ :EXTRA-REQUESTS (APPEND :EXTRA-REQUESTS (MAKE-REQUESTS REQS)


~Replaces the definition of the next word with REQS
(DF CHANGE-NEXT-WORD-SENSE (REQS)
    (SKIP-NEXT-WORD)
    (SETQ :EXTRA-REQUESTS (APPEND :EXTRA-REQUESTS (MAKE-REQUESTS REQS)

~This is the only way to get a request into the :SPECIAL-POOL;
(DF ACTIVATE-LEXICAL-REQS (REQS)
  (PUTPROP @:LEXICAL-POOL
           (APPEND (MAKE-REQUESTS REQS) (GET @:LEXICAL-POOL @REQUESTS))
           @REQUESTS)


~In addition to killing specifically named requests, it can take as special
~arguments the atoms SELF and REST-OF-POOL.
~REQUEST is a variable local to CONSIDER but free here; its value is the name
~of the the request who's actions are presently being evaluated.
~POOL is a variable local to CONSIDER-POOL but free here; its value is the
~name of the request pool currently being considered.
(DF KILL (REQS)
  (MAPC (FUNC (R)
          (COND ((EQ R @SELF) (PUTPROP REQUEST NIL @ACTIVE))
                ((EQ R @REST-OF-POOL)
                 (MAPC '(LAMBDA (X)
                          (PUTPROP X NIL @ACTIVE))
                       (GET POOL @REQUESTS)))
                (T (PUTPROP R NIL @ACTIVE))))
        REQS)


~ADD-DUMMY is used to build an atomized dummy structure and place it on the
~:C-LIST. The structure can then be used as an argument to PRECEDES and
~FOLLOWS, and also ADD-CON;
~:C-LIST is often treated as a stack, so most recent item is at the front.
~.                                              (Changed  WB 11/8/79)
(DE ADD-DUMMY (CON)
  (LET (CON (BUILD-CD @(NIL) NIL NIL))
       (SETQ :C-LIST (CONS CON :C-LIST))
       CON)

~KILL-DUMMY deletes a dummy which was previously created for marking
~purposes
```

```
(DE KILL-DUMMY (DUMMY)
 (SETQ :C-LIST (REMOVE DUMMY :C-LIST))


&
ADD-CON is used to build an atomized CD structure and place it on  the
:C-LIST; it takes four (optional) arguments:
The first must evaluate to a legal CD conceptualization,
 namely the one which is to be atomized and placed on the :C-LIST.
The second must evaluate to a list of pairs of the form (PATH CD);
 the CD will be atomized and placed at the end of the path in
 the structure built by the first argument.
The third argument must evaluate to a list of pairs of the form (PATH1 PATH2);
 the CD at the end of PATH1 replaces the CD at the end of PATH2 in the
 structure built by the first argument.
If there is no fourth argument, then the new CD is placed on the end (=front)
of the :C-LIST.  If there is a fourth argument, then it must evaluate to a
dummy marker on the :C-LIST, which gets its value set to the value of the
new CD.  The value returned is the name of the atom on the :C-LIST.

Note:  (change by MB 11/8/79)
:C-LIST is usu. searched in "most recent first" order, so the conceptual
"end" of the :C-LIST is actually the front of the list for efficiency.
&
(DFX ADD-CON (X)
  (LET (CON NIL
        CONCEPT (EVAL (CAR X))
        FILLERS (EVAL (CADR X))
        EQUIVALENCES (EVAL (CADDR X))
        DUMMY (EVAL (CADDDR X)))
        (SETQ !NEW-CON (BUILD-CD CONCEPT FILLERS EQUIVALENCES))
        (COND (DUMMY (SET DUMMY (EVAL !NEW-CON)) DUMMY)
              (T (SETQ :C-LIST (CONS !NEW-CON :C-LIST))
                 !NEW-CON))


~FILL-GAP puts FILLER at the end of PATH in CD.
~~Since this is usually called
~~when FILLER has been found on the :C-LIST, one of the functions of FILL-GAP
~~is to remove FILLER from the :C-LIST, unless the item is important to memory.
~ items are no longer removed from the :C-LIST, just marked EMBEDDED (MB 11/9)
(DE FILL-GAP (PATH CD FILLER)
        (SET-GAP PATH CD FILLER)  ~ adds path if needed
~ Save the subject or focus of the clause on the act built.
        (COND ((AND (CONCEPT CD) (PRECEDES FILLER CD))
               (PUT CD (REALCON FILLER) @CONTOPIC)))
        (SETQ :LAST-EMBEDDED-CON FILLER)

~   Assume memory wants to know about everything. (see change above)
~       (COND
~        ((OR (FEATURE FILLER 'CONCEPT)  ~memory wants to know about concepts
~             (FEATURE FILLER 'PP)       ~and PP's
~             (FEATURE FILLER 'TIME))    ~and possibly time

          (PMSG 'FILL-GAP T "Embedding " FILLER " in " CD)
~         (PUTPROP (REALCON FILLER) CD 'EMBEDDED))
          (PUTPROP FILLER CD 'EMBEDDED))
~        (T (SETQ :C-LIST (REMOVE FILLER :C-LIST)))

~ shouldn't need this
```

```
~COPY-GAP embeds a copy of the filler in the slot
(DE COPY-GAP (PATH CD FILLER)
  (LET (NEWCON (ADD-CON (ATOM-EVAL FILLER)))
   (PUTPROP NEWCON FILLER 'REALCON)
   (FILL-GAP PATH CD NEWCON)


~SET-GAP puts FILLER at the end of PATH in CD & updates corresponding
~. memory token (if any)
(DE SET-GAP (PATH CD FILLER)
          (SET-ROLE-FILLER PATH CD FILLER)
          (FILL-MEM-GAPS PATH CD FILLER)) ~ fill corresponding gaps in
                                           ~.  memory tokens (&goals,plans)
                                           ~.(this fn in PREDS.LSP)

~MERGE-CONS takes two conceptualizations on the C-list and creates a new
~conceptualization consisting of the union of the semantic content of the
~conceptualizations
(DE MERGE-CONS (CON1 CON2)
    (PMSG 'MERGE-CONS T "Merging " CON2 " into " CON1)
    (SET CON1 (APPEND (ATOM-EVAL CON1) (CDR (ATOM-EVAL CON2))))
~.   (LET (REALCON (REALCON CON1))
~.      (COND (REALCON (SET REALCON (ATOM-EVAL CON1)))
    (SETQ :CHANGED-CONS (REMOVE CON2 :CHANGED-CONS))
    (SETQ :C-LIST (REMOVE CON2 :C-LIST))
```