

*CA4.LSP: W. L. Johnson 10/79 revision of Larry Birnbaum's CA4.MUL 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

(DEF 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;

(DEF ADD-TO-WORD-SENSE (REQS))

(SETQ :EXTRA-REQUESTS (APPEND :EXTRA-REQUESTS (MAKE-REQUESTS REQS)

* Replaces the definition of the next word with REQS

(DEF 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;

(DEF 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 request whose 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.

(DEF KILL (REQS))

(MAPC (FUND (R)

(COND ((EQ R :SELF) (PUTPROP REQUEST NIL :ACTIVE)))

((EQ R :REST-OF-POOL)

(MAPC '(LAMBDA (X)

(PUTPROP X NIL :ACTIVE))

(GET POOL :REQUESTS)))

IF (PUTPROP R NIL :ACTIVE)))

REQS)

* ADD-DUMMY is used to build an atomized dummy structure and place it on the
* C-C-LIST. The structure can then be used as an argument to PRECEDES and
* FOLLOWS, and also ADD-CON;

* C-C-LIST is often treated as a stack, so most recent item is at the front.

(changed MB 11/8/79)

(DEF ADD-DUMMY (CON)

(LET (CON (BUILD-CD :C(NIL) NIL NIL))

(SETQ :C-LIST (CONS CON :C-LIST))

CON)

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

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

%
 ADD-CON is used to build an atomized CO structure and place it on the :C-LIST; it takes four (optional) arguments:
 The first must evaluate to a legal CO 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.

```
(DEF ADD-CON (X)
  (LET (CON WITH
        CONCEPT (EVAL (CAR X))
        FILLERS (EVAL (CADR X))
        EQUIVALENCES (EVAL (CAADR X))
        DUMMY (EVAL (CADDR X)))
    (SETQ !NEW-CON (BUILD-CO 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 CO.
 *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)

```
(DEF 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) 'CONTDPICD))
        (T (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
      (PMSSG '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)
  (DEF LNERCON (ADD-CON (ATOM-EVAL FILLER))
    (POPPROP NEWCON FILLER 'REACCON)
    (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 PRED5.LSP)

```

```

MERGE-CONS takes two conceptualizations on the D-list and creates a new
conceptualization consisting of the union of the semantic content of the
conceptualizations

```

```

(DE MERGE-CONS (CON1 CON2)

```

```

  (PUSH MERGE-CONS T "Merging " CON2 " into " CON1)

```

```

  (SET CON1 (APPEND (ATOM-EVAL CON1) (CDR (ATOM-EVAL CON2))))

```

```

  (DEF (REACCON (REACCON CON1))

```

```

    (COND (REACCON (SET REACCON (ATOM-EVAL CON1)))

```

```

    (SET) :CHANGED-CONS (REMOVE CON2 :CHANGED-CONS))

```

```

  (SET) :C-DIST (REMOVE CON2 :C-DIST)

```