```
;This is the file CAS.LSP; it contains the functions for building and
;manipulating CDs;
;10/79 MBJ Revision of Burnbaum 8/79


%
BUILD-CD is used to build an atomized CD structure; it takes three
arguments; the first must evaluate to a legal CD conceptualization,
namely the one which is to be atomized; 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 PATH1 is
NIL, then the entire concept is returned, so that back-pointers can be set;
%
(DE BUILD-CD (CONCEPT FILLERS EQUIVALENCES)
  (LET (CON (MAKE-CD CONCEPT))
       (MAPC (FUNCTION (LAMBDA (E)
                       (SUBST-CD (GET-ROLE-FILLER (CAR E) CON)
                                 (GET-ROLE-FILLER (CADR E) CON)
                                 CON))))
             EQUIVALENCES)
       (MAPC (FUNCTION (LAMBDA (F)
                       (SET-ROLE-FILLER (CAR F) CON (CADR F))))
             FILLERS)
       CON))


;GET-ROLE-VALUE is like GET-ROLE-FILLER, except that if the role-filler
;to be returned has been atomized, it is EVALed first;
(DE GET-ROLE-VALUE (PATH CONCEPT)
  (ATOM-EVAL (GET-ROLE-FILLER PATH CONCEPT)))


;GET-ROLE-FILLER returns the end of the path which it's first argument
;evaluates to within the CD which its second argument evaluates to;
(DE GET-ROLE-FILLER (PATH CONCEPT)
  (COND ((NULL PATH) CONCEPT) (T (NPATH CONCEPT PATH))))


;SET-ROLE-FILLER puts an atomized form of its third argument at the end
;of the path specified by its first argument within its second argument;
;it does this by actually replacing the atom, NOT simply by setting the
;value;
;; If slot isn't there, it is added. (changed 11/11/79 MB)
(DEX SET-ROLE-FILLER (PATH CONCEPT FILLER)
  (LET (NC (MAKE-CD FILLER)
       OC (GET-ROLE-FILLER PATH CONCEPT))
    (COND (OC (SUBST-CD NC OC CONCEPT)
              (CHAIN-GAP OC NC))          ;IN PREDS.LSP (propagate slot change)
          (T (ADD-GAP PATH CONCEPT NC)))
    (COND ((IS-RUNNING CA)
           (LIST-ADD CONCEPT a:CHANGED-CONS)))
    (PUTPROP NC CONCEPT 'EMBEDDED)
    CONCEPT))


;This adds a gap to a CD at the end of PATH; only works if all but last role
;in PATH already exists, and the last ROLE is to be added. Filler is FILLER.
```

```
(DE ADD-GAP (PATH CONCEPT FILLER)
  (LET (CON (GET-ROLE-FILLER (BUTLAST PATH) CONCEPT) ROLE (LAST PATH))
    (COND ((AND CON (NOT (GET-ROLE-FILLER ROLE CON)))
           (SET CON (APPEND (ATOM-EVAL CON)
                            (LIST (CAR ROLE) (MAKE-CD FILLER))))
           ROLE)
          (T NIL))))        ;else an error?


;MAKE-CD atomizes a conceptualization unless it already is atomized;
;it also fires any trigger demons attached to the new concept;
(DEX MAKE-CD (X)
  (LET (CON (BUILD-C X NIL))
       (COND ((NEQ X CON)
              (FIRE-DEMONS CON (HAS-VALUE @DUMMY))
              (AND (IS-RUNNING CA)
                   (LIST-ADD (OR (HAS-VALUE @DUMMY) CON)
                             @:CHANGED-CONSJ)))
       CON))


;BUILD-C and BUILD-M call each other recursively to atomize a CD;
(DE BUILD-C (X SEEN)

  (COND ((ATOM X) X)
        ((EQUAL X @(PREVIOUS)) (CADR SEEN))
        ((PROG (NEWCON)
;In case someone passes an XPNed CD, this sets the back-pointer properly.
              (SETQ NEWCON (NEW-CON))
              (SETQ SEEN (CONS NEWCON SEEN))
              (SET NEWCON (CONS (BUILD-M (CAR X) SEEN) (BUILD-M (CDR X) SEEN)))
              (RETURN NEWCON)))))


;BUILD-C and BUILD-M call each other recursively to atomize a CD;
(DE BUILD-M (X SEEN)
  (PROG (TEMP)
        (AND (ATOM X) (RETURN X))
        (SETQ TEMP (LIST NIL))
   LOOP (NCONC TEMP (LIST (CAR X) (BUILD-C (CADR X) SEEN)))
        (COND ((NULL (SETQ X (CDDR X))) (RETURN (CDR TEMP)))
              ((NULL (CDR X))
               (MSG T "BAD MODIFIER LIST" X T)
               (ERR NIL))
              (T (GO LOOP)))))


;SUBST-CD replaces all occurrences of its second argument with its first
;argument in its third argument; it assumes its arguments are atomized CDs;
(DE SUBST-CD (NEW-C OLD-C CD)
  (SUBCDC NEW-C OLD-C CD NIL))


;SUBCDC and SUBCDM call each other recursively to accomplish SUBST-CD;
(DE SUBCDC (NEW-C OLD-C CD SEEN)
   (COND ((EQ CD OLD-C) NEW-C)
         ((MEMQ CD SEEN) CD)
         (T
```

```
        (SETQ SEEN (CONS CD SEEN))
        (SET CD (CONS (SUBCDM NEW-C OLD-C (CAR (EVAL CD)) SEEN)
                      (SUBCDM NEW-C OLD-C (CDR (EVAL CD)) SEEN)))
        CD))


;SUBCDC and SUBCDM call each other recursively to accomplish SUBST-CD;
(DE SUBCDM (NEW-C OLD-C FORM SEEN)
  (PROG (TEMP)
        (COND ((EQ FORM OLD-C) (RETURN NEW-C))
              ((ATOM FORM) (RETURN FORM)))
        (SETQ TEMP (LIST NIL))
   LOOP (NCONC TEMP
               (LIST (CAR FORM) (SUBCDC NEW-C OLD-C (CADR FORM) SEEN)))
        (COND ((SETQ FORM (CDDR FORM)) (GO LOOP))
              (T (RETURN (CDR TEMP))))))
```