

File created: 20-Dec-88 09:27:17 {DSK}<LISPPFILES>LOGIC>MEDLEY>LOGIC.;3

changes to: (IL:VARS IL:LOGICCOMS)  
(IL:FUNCTIONS CREATE-BACKGROUND-THEORY SHOW-THEORY)

previous date: 19-Dec-88 10:50:29 {DSK}<LISPPFILES>LOGIC>MEDLEY>LOGIC.;2

Read Table: INTERLISP

Package: USER

Format: XCCS

::  
:: Copyright (c) 1988 by Roberto Ghislanzoni. All rights reserved.

(IL:RPAQQ **IL:LOGICCOMS**

```
((IL:* IL:THESE IL:ARE IL:MACROS)
 (IL:FUNCTIONS AND-LEVEL ANTEC ATOMIC-FORMULAP CLAUSES-OR CONJ CONSEQP DIRECTLY-IMPLEMENTED FAILEDP
  FORMULA-OR GET-AND-NODE-THEORIES GET-CUT GET-OR-NODE-THEORIES GET-THEORY IMPLICATIONP
  NULL-AND-LEVELP NULL-OR-LEVELP NULL-TREEP OR-LEVELS SEMANTIC-ATTACHMENT-P THEORYP UNIF-ENV-OR
  UNIFICATION-ENV)
 (IL:* AND IL:THESE IL:ARE IL:FUNCTIONS)
 (IL:FUNCTIONS ADD-OR-LEVEL ALL ALL-PREDICATES ALL-PREDS ALL-SAS ALL-SEMANTIC-ATTACHMENTS ANY ATTACH
  CLEAR-AND-LEVEL CONSEQ CREATE-BACKGROUND-THEORY CREATE-THEORY DELETE-OR-NODE
  DELETE-OR-NODE-WITH-CUT FIND-CLAUSES IS-THERE-CUT LIST-ALL-THEORIES LOAD-THEORY LOGIC-ADDA
  LOGIC-ADDZ LOGIC-ASSERT LOGIC-DELETE LOGIC-DELETE-FACT LOGIC-PROVE MAKE-AND-NODE MAKE-OR-NODE
  MAKE-TREE MERGE-INTERNAL MERGE-THEORIES NEW-TREE PREDICATE PROVE RENAME-CUT SAVE-THEORY
  SHOW-DEFINITION SHOW-THEORY SOLVE SUBSTITUTE-LEVEL UPDATE-ENV UPDATE-LEVEL UPDATE-TREE)
 (IL:VARS *PRINT-PRETTY*)
 (IL:P (IL:FILESLOAD LOGIC-UNIFIER))))
```

(IL:\* IL:\* IL:THESE IL:ARE IL:MACROS)

(DEFMACRO **AND-LEVEL** (TREE)  
 `(CAR ,TREE))

(DEFMACRO **ANTEC** (WFF)  
 `(CDDR ,WFF))

(DEFMACRO **ATOMIC-FORMULAP** (WFF)  
 `[AND (LISTP ,WFF)  
 (NULL (SECOND ,WFF])

(DEFMACRO **CLAUSES-OR** (OR-NODE)  
 `(SECOND ,OR-NODE))

(DEFMACRO **CONJ** (AND-LEVEL)  
 `(CAR ,AND-LEVEL))

(DEFMACRO **CONSEQP** (C)  
 `[AND (LISTP ,C)  
 (SYMBOLP (CAR ,C))

(DEFMACRO **DIRECTLY-IMPLEMENTED** (CLAUSES)  
 `(EQ (CAR ,CLAUSES)  
 'DIRECTLY-IMPLEMENTED))

(DEFMACRO **FAILEDP** (ENV)  
 `(EQ ,ENV 'FAILED))

(DEFMACRO **FORMULA-OR** (OR-LEVEL)  
 `(CAR ,OR-LEVEL))

(DEFMACRO **GET-AND-NODE-THEORIES** (AND-NODE)  
 `(THIRD ,AND-NODE))

(DEFMACRO **GET-CUT** (OR-NODE)  
 `(SIXTH ,OR-NODE))

(DEFMACRO **GET-OR-NODE-THEORIES** (OR-NODE)  
 `(FIFTH ,OR-NODE))

```
(DEFMACRO GET-THEORY (THEORY-NAME &OPTIONAL WINDOW)
  `(OR (AND ,WINDOW (GET-THEORY-INTERNAL ,THEORY-NAME ,WINDOW))
      (GET 'THEORY ,THEORY-NAME)))
```

```
(DEFMACRO IMPLICATIONP (WFF)
  `[LET [(SEPARATOR (SECOND ,WFF))
        (AND (EQ SEPARATOR ':-)
              (NOT (NULL (CDDR ,WFF)))]
```

```
(DEFMACRO NULL-AND-LEVELP (TREE)
  `(NULL (CAR ,TREE)))
```

```
(DEFMACRO NULL-OR-LEVELP (TREE)
  `(NULL (SECOND ,TREE)))
```

```
(DEFMACRO NULL-TREEP (TREE)
  `(AND (NULL-AND-LEVELP ,TREE)
        (NULL-OR-LEVELP ,TREE)))
```

```
(DEFMACRO OR-LEVELS (TREE)
  `(SECOND ,TREE))
```

```
(DEFMACRO SEMANTIC-ATTACHMENT-P (SA)
  `(EQ (CAR ,SA)
      'SA))
```

```
(DEFMACRO THEORYP (THEORY &OPTIONAL WINDOW)
  `(OR (AND (GET-THEORY ,THEORY ,WINDOW)
            T)
      (HASH-TABLE-P ,THEORY)))
```

```
(DEFMACRO UNIF-ENV-OR (OR-NODE)
  `(FOURTH ,OR-NODE))
```

```
(DEFMACRO UNIFICATION-ENV (AND-NODE)
  `(SECOND ,AND-NODE))
```

(IL:\* IL:\* AND IL:THESE IL:ARE IL:FUNCTIONS)

```
(DEFUN ADD-OR-LEVEL (WFF CLAUSES TREE &OPTIONAL CUTNAME)
  ;; Adds a new or-node to the list of the nodes. The new node is put in front of the old ones
  [COND
   ((NULL CLAUSES)
    TREE)
   (T (LET* ((LEVEL (AND-LEVEL TREE))
             (NEW-OR-NODE (MAKE-OR-NODE WFF CLAUSES (CONJ LEVEL)
                                       (UNIFICATION-ENV LEVEL)
                                       (GET-AND-NODE-THEORIES LEVEL)
                                       CUTNAME)))
          (MAKE-TREE LEVEL (APPEND (LIST NEW-OR-NODE)
                                     (OR-LEVELS TREE)))]
```

```
(DEFUN ALL (VARS CONJ THS)
  [PROG (RESULTING-TREE (*VARIABLES-COUNTER* 0)
        (TREE (MAKE-TREE (MAKE-AND-NODE CONJ NIL (APPEND (LIST '*BACKGROUND-THEORY*)
                                                             THS))
                        NIL))
        COLLECTED-RESULTS NEXT-OR)
  (DECLARE (SPECIAL *VARIABLES-COUNTER*))
  HERE
  (SETF RESULTING-TREE (LOGIC-PROVE TREE))
  (COND
   ((NULL RESULTING-TREE)
    (RETURN COLLECTED-RESULTS))
   (T [SETF COLLECTED-RESULTS (APPEND COLLECTED-RESULTS (LIST (LOOKUP VARS (UNIFICATION-ENV
                                                                    (AND-LEVEL RESULTING-TREE))
                                                                    (SETF NEXT-OR (FIRST (OR-LEVELS RESULTING-TREE)))
                                                                    (SETF TREE (SOLVE (NEW-TREE RESULTING-TREE NEXT-OR)
                                                                    (FORMULA-OR NEXT-OR)
                                                                    (CLAUSES-OR NEXT-OR))))
    (GO HERE)]])
```

```
(DEFUN ALL-PREDICATES (THEORY-NAME)
  (ALL-PREDS (GET-THEORY THEORY-NAME)))
```

```
(DEFUN ALL-PREDS (THEORY)
  ;; The presence of VAL in the AND body is necessary because it is correct to test if the predicates has not been erased: in such a case its value is
  ;; NIL
  (PROG (PRNAMES)
    LABEL
      (MAPHASH #'[LAMBDA (KEY VAL)
                (AND (NOT (SEMANTIC-ATTACHMENT-P VAL))
                     VAL
                     (SETF PRNAMES (APPEND PRNAMES (LIST KEY]
                                     THEORY)
                     (RETURN PRNAMES)))
```

```
(DEFUN ALL-SAS (THEORY)
  (PROG (SANAMES)
    LABEL
      (MAPHASH #'[LAMBDA (KEY VAL)
                (AND (SEMANTIC-ATTACHMENT-P VAL)
                     VAL
                     (SETF SANAMES (APPEND SANAMES (LIST KEY]
                                     THEORY)
                     (RETURN SANAMES)))
```

```
(DEFUN ALL-SEMANTIC-ATTACHMENTS (THEORY-NAME)
  (ALL-SAS (GET-THEORY THEORY-NAME)))
```

```
(DEFUN ANY (HOW-MANY VARS CONJ THS)
  [PROG (RESULTING-TREE (*VARIABLES-COUNTER* 0)
        (COUNTER 0)
        (TREE (MAKE-TREE (MAKE-AND-NODE CONJ NIL (APPEND (LIST '*BACKGROUND-THEORY*
                                                                THS))
                        NIL))
              COLLECTED-RESULTS NEXT-OR)
  (DECLARE (SPECIAL *VARIABLES-COUNTER*))
  HERE
  (SETF RESULTING-TREE (LOGIC-PROVE TREE))
  (COND
   ((OR (NULL RESULTING-TREE)
        (EQ COUNTER HOW-MANY))
    (RETURN COLLECTED-RESULTS))
   (T [SETF COLLECTED-RESULTS (APPEND COLLECTED-RESULTS (LIST (LOOKUP VARS (UNIFICATION-ENV
                                                                    (AND-LEVEL RESULTING-TREE))
                                                                (SETF NEXT-OR (FIRST (OR-LEVELS RESULTING-TREE)))
                                                                (SETF TREE (SOLVE (NEW-TREE RESULTING-TREE NEXT-OR)
                                                                    (FORMULA-OR NEXT-OR)
                                                                    (CLAUSES-OR NEXT-OR)))
                                                                (INCF COUNTER)
                                                                (GO HERE])
```

```
(DEFUN ATTACH (SA-NAME DEFINITION THEORY-NAME &OPTIONAL WINDOW)
  (SETF (GETHASH SA-NAME (GET-THEORY THEORY-NAME WINDOW))
        (CONS 'SA DEFINITION)
        'ATTACHED)
```

```
(DEFUN CLEAR-AND-LEVEL (TREE)
  (PROGN (SETF (CAR TREE)
              NIL)
        TREE))
```

```
(DEFUN CONSEQ (WFF)
  (CAR WFF))
```

```
(DEFUN CREATE-BACKGROUND-THEORY ()
  [PROGN (IN-PACKAGE 'USER)
        (CREATE-THEORY '*BACKGROUND-THEORY*)
        (WITH-OPEN-FILE (FILE (MERGE-PATHNAMES (MAKE-PATHNAME :NAME 'LOGIC :TYPE 'LGC))
                          :DIRECTION :INPUT)
          (PROG (NAME)
            LABEL
              (AND (EQ (SETF NAME (READ FILE))
                      'THEORY-END)
                   (RETURN))
              (LOGIC-ASSERT NAME (CONS 'DIRECTLY-IMPLEMENTED (READ FILE))
                              '*BACKGROUND-THEORY*)
              (GO LABEL])
```

```
(DEFUN CREATE-THEORY (THEORY-NAME)
  (SETF (GET 'THEORY THEORY-NAME)
        (MAKE-HASH-TABLE)
        THEORY-NAME)
```

```
(DEFUN DELETE-OR-NODE (TAGNODE NODES)
  (DELETE TAGNODE NODES :TEST #'EQUAL :COUNT 1))
```

```
(DEFUN DELETE-OR-NODE-WITH-CUT (CUTNAME OR-LEVELS)
  ;; This function is called every time a cut is proven: all the alternatives for that clause MUST be erased. Remember that every cut has a unique
  ;; identifier
  [PROG ((NODES OR-LEVELS))
    LABEL
    (COND
      ((NULL NODES)
       (RETURN OR-LEVELS))
      ((EQ (GET-CUT (CAR NODES))
            CUTNAME)
       (RETURN (DELETE-OR-NODE (CAR NODES)
                                 OR-LEVELS)))
      (T (SETF NODES (CDR NODES))
         (GO LABEL])
```

```
(DEFUN FIND-CLAUSES (PREDICATE-NAME THEORY-NAMES &OPTIONAL WINDOW)
  [PROG NIL
    LABEL
    (COND
      ((NULL THEORY-NAMES)
       (RETURN NIL))
      (T (LET* ((TH (FIRST THEORY-NAMES))
                (CLAUSES (BINDING PREDICATE-NAME TH WINDOW)))
          (COND
            ((NULL CLAUSES)
             (SETF THEORY-NAMES (CDR THEORY-NAMES))
             (GO LABEL))
            (T (RETURN CLAUSES])
```

```
(DEFUN IS-THERE-CUT (CONJS)
  [OR (MEMBER '! CONJS)
    (PROG ((ELTS CONJS))
      LABEL
      (COND
        ((NULL ELTS)
         NIL)
        ((AND (SYMBOLP (CAR ELTS))
              (EQ (CHAR-CODE (CHAR (SYMBOL-NAME (CAR ELTS)))
                       0))
              33))
        (RETURN T))
      (T (SETF ELTS (CDR ELTS))
         (GO LABEL])
```

```
(DEFUN LIST-ALL-THEORIES (&OPTIONAL WINDOW)
  [OR (AND WINDOW (LIST-ALL-THEORIES-INTERNAL WINDOW))
    (DO ((LL (SYMBOL-PLIST 'THEORY)
              (CDDR LL))
          (RESULT NIL))
        ((NULL LL)
         RESULT)
      [SETF RESULT (APPEND RESULT (LIST (CAR LL)])])
```

```
(DEFUN LOAD-THEORY (THEORY-NAME &OPTIONAL WINDOW)
  [LET [(THEORY-FILE (MERGE-PATHNAMES (MAKE-PATHNAME :NAME THEORY-NAME :TYPE 'LGC)
                                     (LOAD-DEVEL-THEORY WINDOW THEORY-NAME))
        (OR [AND (PROBE-FILE THEORY-FILE)
                (WITH-OPEN-FILE (FILE THEORY-FILE :DIRECTION :INPUT)
                              (PROG (THEORY-NAME PRED-NUMBER SAS-NUMBER)
                                (SETF THEORY-NAME (READ FILE))
                                (CREATE-THEORY THEORY-NAME)
                                (SETF SAS-NUMBER (READ FILE))
                                (DO ((SAS SAS-NUMBER (DECF SAS))
                                    (EQ SAS 0)
                                    NIL)
                                  (SETF (GETHASH (READ FILE)
                                                  (GET 'THEORY THEORY-NAME))
                                        (READ FILE)))
                                (SETF PRED-NUMBER (READ FILE))
                                (DO ((PREDS PRED-NUMBER (DECF PREDS))
                                    (EQ PREDS 0)
```

```

      NIL)
      (SETF (GETHASH (READ FILE)
                    (GET 'THEORY THEORY-NAME))
            (READ FILE)))
      (RETURN 'LOADED])
(FORMAT T "Theory not found"])
```

```
(DEFUN LOGIC-ADDA (PRED CLAUSES THEORY &OPTIONAL WINDOW)
  (PROGN [SETF (GETHASH PRED (GET-THEORY THEORY WINDOW))
              (APPEND CLAUSES (GETHASH PRED (GET-THEORY THEORY WINDOW)
                                          'ADDED))])
```

```
(DEFUN LOGIC-ADDZ (PRED CLAUSES THEORY &OPTIONAL WINDOW)
  (PROGN (SETF (GETHASH PRED (GET-THEORY THEORY WINDOW))
              (APPEND (GETHASH PRED (GET-THEORY THEORY WINDOW)
                                CLAUSES)
                      'ADDED)))
```

```
(DEFUN LOGIC-ASSERT (PREDICATE-NAME CLAUSES THEORY-NAME &OPTIONAL WINDOW)
  (SETF (GETHASH PREDICATE-NAME (GET-THEORY THEORY-NAME WINDOW))
        CLAUSES)
  'ASSERTED)
```

```
(DEFUN LOGIC-DELETE (PRED-OR-SA THEORY-NAME &OPTIONAL WINDOW)
  (PROGN (SETF (GETHASH PRED-OR-SA (GET-THEORY THEORY-NAME WINDOW))
              NIL)
        'DELETED))
```

```
(DEFUN LOGIC-DELETE-FACT (FACT-NAME FACT-CLAUSE THEORY &OPTIONAL WINDOW)
  ;; deletes from the definition of facts one of the definitions themselves
  ;; ((ON a b) (ON b c)) --> ((ON a b))
  (PROGN (SETF (GETHASH FACT-NAME (GET-THEORY THEORY WINDOW))
              (DELETE FACT-CLAUSE (GETHASH FACT-NAME (GET-THEORY THEORY WINDOW))
                          :TEST
                          #'EQUAL))
        'DELETED))
```

```
(DEFUN LOGIC-PROVE (TREE &OPTIONAL WINDOW)
  [PROG ((*VARIABLES-COUNTER* -1))
  (DECLARE (SPECIAL *VARIABLES-COUNTER*))
  ;; This is a counter for the variables that will be used during the unification
  JUMP
  (COND
    ((NULL-TREEP TREE)
     (RETURN NIL))
    [(NULL-AND-LEVELP TREE)
     (LET [(NEXT-OR (FIRST (OR-LEVELS TREE))
                       ;; Gets the next or-node: we have now no strategy for choosing it; maybe later...
          (COND
            ((NULL NEXT-OR)
             (SETF TREE (LIST NIL NIL))
             (GO JUMP))
            (T (SETF TREE (SOLVE (NEW-TREE TREE NEXT-OR)
                                (FORMULA-OR NEXT-OR)
                                (CLAUSES-OR NEXT-OR)
                                NIL WINDOW))
                (GO JUMP]
          (T (LET ((NEXT-LEVEL (AND-LEVEL TREE)))
              (COND
                ((NULL (CONJ NEXT-LEVEL))
                 (RETURN TREE))
                (T (LET* [(TO-PROVE (FIRST (CONJ NEXT-LEVEL)))
                        (THS (GET-AND-NODE-THEORIES NEXT-LEVEL))
                        (CLAUSES (FIND-CLAUSES (PREDICATE TO-PROVE)
                                              THS WINDOW))
                        (CUT? (IS-THERE-CUT (REST (CONJ NEXT-LEVEL))
                                           (SETF TREE (SOLVE (UPDATE-TREE (UPDATE-LEVEL NEXT-LEVEL TO-PROVE)
                                                                    TREE)
                                                                TO-PROVE CLAUSES CUT? WINDOW))
                        (GO JUMP]))])
```

```
(DEFUN MAKE-AND-NODE (CONJ UNIF-ENV THEORIES)
  (LIST CONJ UNIF-ENV THEORIES))
```

```
(DEFUN MAKE-OR-NODE (WFF CLAUSES BORDER UNIF-ENV THEORIES &OPTIONAL CUTNAME)
  (LIST WFF CLAUSES BORDER UNIF-ENV THEORIES CUTNAME))
```



```

      (FORMAT FILE "~D~%" (LENGTH PRED))
      (DO ((PRED-NAME PRED (CDR PRED-NAME))
          ((NULL PRED-NAME)
           NIL)
          (FORMAT FILE "~S ~S ~%" (CAR PRED-NAME)
            (GETHASH (CAR PRED-NAME)
              THEORY)))
        'SAVED])

```

```

(DEFUN SHOW-DEFINITION (ELEMENT THEORY-NAME &OPTIONAL WINDOW)
  [FORMAT (OR (AND WINDOW *TRACE-OUTPUT*
    T)
    "~S~%"
    (PROG [(DEF (GETHASH ELEMENT (GET-THEORY THEORY-NAME WINDOW)
      (OR (AND (SEMANTIC-ATTACHMENT-P DEF)
        (RETURN (CDR DEF)))
        (RETURN DEF))

```

```

(DEFUN SHOW-THEORY (THEORY-NAME &OPTIONAL VERBOSE WINDOW)
  [LET* ((THEORY (GET-THEORY THEORY-NAME))
    (PREDICATES (SORT (ALL-PREDS THEORY)
      #'STRING-LESSP))
    (SAS (SORT (ALL-SAS THEORY)
      #'STRING-LESSP))
    (STREAM (OR (AND WINDOW *TRACE-OUTPUT*
      T)))
    [OR (AND SAS (PROGN (FORMAT STREAM "Semantic attachments: ~%"
      (DO ((PP SAS (CDR PP))
        ((NULL PP)
         NIL)
        (PROGN (FORMAT T "~%~S ~%" (CAR PP)
          (AND VERBOSE (FORMAT T "Definition: ~S ~%" (CDR (GETHASH (CAR PP)
            THEORY))
              " "))))
      (FORMAT STREAM "~% ~%"
    (OR (AND PREDICATES (PROGN (FORMAT STREAM "Predicates: ~%"
      (DO ((PP PREDICATES (CDR PP))
        ((NULL PP)
         NIL)
        (PROGN (FORMAT T "~%~S ~%" (CAR PP)
          (AND VERBOSE (FORMAT STREAM "Clauses: ~S ~%" (GETHASH (CAR PP)
            THEORY)
              " "))))
      (FORMAT STREAM "~% ~%"

```

```

(DEFUN SOLVE (TREE FORMULA CLAUSES &OPTIONAL CUT WINDOW)
  [PROG NIL
    JUMP
    (AND WINDOW (SOLVE-DEBUGGER TREE FORMULA CLAUSES WINDOW))
    (COND
      ((NULL CLAUSES) ; demo is failed
        (RETURN (CLEAR-AND-LEVEL TREE)))
      ((DIRECTLY-IMPLEMENTED CLAUSES) ; clauses from the main theory
        (RETURN (FUNCALL (CDR CLAUSES)
          TREE FORMULA CLAUSES WINDOW)))
      [(SEMANTIC-ATTACHMENT-P CLAUSES) ; Semantic attachment defined by the user
        (LET [(EXPANDED-FORMULA (LOOKUP FORMULA (UNIFICATION-ENV (AND-LEVEL TREE)
          (COND
            ((APPLY (CDR CLAUSES)
              (CDR EXPANDED-FORMULA))
              (RETURN TREE))
            (T (RETURN (CLEAR-AND-LEVEL TREE]
          (T (LET* ((CANDIDATE (FIRST CLAUSES))
            (ASSERT (RENAME CANDIDATE))
            (NEWENV (UNIFY FORMULA (CONSEQ ASSERT)
              (UNIFICATION-ENV (AND-LEVEL TREE))
              WINDOW)))
            (COND
              ((FAILEDPP NEWENV)
                (SETF CLAUSES (REST CLAUSES))
                (GO JUMP))
              [(ATOMIC-FORMULAP ASSERT)
                ;; If a cut has been discovered in the previous procedure, it is necessary not to instantiate alternatives for the clause in
                ;; a or-level
                (RETURN (UPDATE-ENV NEWENV (OR (AND CUT TREE)
                  (ADD-OR-LEVEL FORMULA (REST CLAUSES)
                    TREE]
                ((IMPLICATIONP ASSERT)
                  ;; If there is a cut, it is necessary to mark the alternatives for that clause; if the cut will be proved, then these
                  ;; alternatives will be eliminated
                  (RETURN (COND

```

```
[ (IS-THERE-CUT (ANTEC ASSERT))
  (LET* ((RENAMED-STRUCTURE (RENAME-CUT (ANTEC ASSERT)))
         (RENAMED-CUT (CAR RENAMED-STRUCTURE))
         (RENAMED-ASSERT (CDR RENAMED-STRUCTURE)))
    (SUBSTITUTE-LEVEL NEWENV RENAMED-ASSERT (ADD-OR-LEVEL FORMULA
                                                       (REST CLAUSES)
                                                       TREE RENAMED-CUT])
    (T (SUBSTITUTE-LEVEL NEWENV (ANTEC ASSERT)
                               (ADD-OR-LEVEL FORMULA (REST CLAUSES)
                                                  TREE]))
```

```
(DEFUN SUBSTITUTE-LEVEL (ENV ANTECS TREE)
  (PROGN [RPLACA TREE (MAKE-AND-NODE (APPEND ANTECS (CONJ (AND-LEVEL TREE)))
                                     ENV
                                     (GET-AND-NODE-THEORIES (AND-LEVEL TREE]
        TREE))
```

```
(DEFUN UPDATE-ENV (ENV TREE)
  (SETF (SECOND (AND-LEVEL TREE))
        ENV)
  TREE)
```

```
(DEFUN UPDATE-LEVEL (LEVEL FORMULA)
  (MAKE-AND-NODE (CDR (CONJ LEVEL))
                (UNIFICATION-ENV LEVEL)
                (GET-AND-NODE-THEORIES LEVEL)))
```

```
(DEFUN UPDATE-TREE (LEVEL TREE)
  (MAKE-TREE LEVEL (OR-LEVELS TREE)))
```

```
(IL:RPAQQ *PRINT-PRETTY* T)
```

```
(IL:FILESLOAD LOGIC-UNIFIER)
```

```
(IL:PUTPROPS IL:LOGIC IL:COPYRIGHT ("Roberto Ghislanzoni" 1988))
```



---

**FUNCTION INDEX**

ADD-OR-LEVEL . . . . .	2	CREATE-THEORY . . . . .	4	LOGIC-DELETE-FACT . . . . .	5	SAVE-THEORY . . . . .	6
ALL . . . . .	2	DELETE-OR-NODE . . . . .	4	LOGIC-PROVE . . . . .	5	SHOW-DEFINITION . . . . .	7
ALL-PREDICATES . . . . .	3	DELETE-OR-NODE-WITH-CUT . . . . .	4	MAKE-AND-NODE . . . . .	5	SHOW-THEORY . . . . .	7
ALL-PREDS . . . . .	3	FIND-CLAUSES . . . . .	4	MAKE-OR-NODE . . . . .	5	SOLVE . . . . .	7
ALL-SAS . . . . .	3	IS-THERE-CUT . . . . .	4	MAKE-TREE . . . . .	6	SUBSTITUTE-LEVEL . . . . .	8
ALL-SEMANTIC-ATTACHMENTS . . . . .	3	LIST-ALL-THEORIES . . . . .	4	MERGE-INTERNAL . . . . .	6	UPDATE-ENV . . . . .	8
ANY . . . . .	3	LOAD-THEORY . . . . .	4	MERGE-THEORIES . . . . .	6	UPDATE-LEVEL . . . . .	8
ATTACH . . . . .	3	LOGIC-ADDA . . . . .	5	NEW-TREE . . . . .	6	UPDATE-TREE . . . . .	8
CLEAR-AND-LEVEL . . . . .	3	LOGIC-ADDZ . . . . .	5	PREDICATE . . . . .	6		
CONSEQ . . . . .	3	LOGIC-ASSERT . . . . .	5	PROVE . . . . .	6		
CREATE-BACKGROUND-THEORY . . . . .	3	LOGIC-DELETE . . . . .	5	RENAME-CUT . . . . .	6		

---

**MACRO INDEX**

AND-LEVEL . . . . .	1	DIRECTLY-IMPLEMENTED . . . . .	1	GET-THEORY . . . . .	2	SEMANTIC-ATTACHMENT-P . . . . .	2
ANTEC . . . . .	1	FAILEDP . . . . .	1	IMPLICATIONP . . . . .	2	THEORYP . . . . .	2
ATOMIC-FORMULAP . . . . .	1	FORMULA-OR . . . . .	1	NULL-AND-LEVELP . . . . .	2	UNIF-ENV-OR . . . . .	2
CLAUSES-OR . . . . .	1	GET-AND-NODE-THEORIES . . . . .	1	NULL-OR-LEVELP . . . . .	2	UNIFICATION-ENV . . . . .	2
CONJ . . . . .	1	GET-CUT . . . . .	1	NULL-TREEP . . . . .	2		
CONSEQP . . . . .	1	GET-OR-NODE-THEORIES . . . . .	1	OR-LEVELS . . . . .	2		

---

**VARIABLE INDEX**

*PRINT-PRETTY* . . . . .	8
--------------------------	---

---