

File created: 18-May-90 01:28:45 {DSK}<usr>local>lde>lispcore>sources>XCLC-ANNOTATE.;2

changes to: (IL:VARS IL:XCLC-ANNOTATECOMS)

previous date: 3-May-88 17:43:47 {DSK}<usr>local>lde>lispcore>sources>XCLC-ANNOTATE.;1

Read Table: XCL

Package: COMPILER

Format: XCCS

; Copyright (c) 1986, 1987, 1988, 1990 by Venue & Xerox Corporation. All rights reserved.

(IL:RPAQQ **IL:XCLC-ANNOTATECOMS**

(;; Annotation of the program tree

(IL:FUNCTIONS ANNOTATE-TREE)

;; Frame Annotation

(IL:VARIABLES *REFERENCES*)

(IL:FUNCTIONS FRAME-ANNOTATE CLOSE-OVER)

(IL:FUNCTIONS FRAME-ANNOTATE-BLOCK FRAME-ANNOTATE-CALL FRAME-ANNOTATE-CATCH FRAME-ANNOTATE-GO
FRAME-ANNOTATE-IF FRAME-ANNOTATE-LABELS FRAME-ANNOTATE-LAMBDA FRAME-ANNOTATE-LITERAL
FRAME-ANNOTATE-MV-CALL FRAME-ANNOTATE-MV-PROG1 FRAME-ANNOTATE-OPCODES FRAME-ANNOTATE-PROGN
FRAME-ANNOTATE-PROGV FRAME-ANNOTATE-RETURN FRAME-ANNOTATE-SETQ FRAME-ANNOTATE-TAGBODY
FRAME-ANNOTATE-THROW FRAME-ANNOTATE-UNWIND-PROTECT FRAME-ANNOTATE-VAR-REF)

;; Closure annotation

(IL:VARIABLES *NEED-STORAGE*)

(IL:FUNCTIONS CLOSURE-ANNOTATE)

(IL:I.S.OPRS UNIONING)

(IL:FUNCTIONS CLOSURE-ANNOTATE-BLOCK CLOSURE-ANNOTATE-CALL CLOSURE-ANNOTATE-CATCH CLOSURE-ANNOTATE-GO
CLOSURE-ANNOTATE-IF CLOSURE-ANNOTATE-LABELS CLOSURE-ANNOTATE-LAMBDA CLOSURE-ANNOTATE-LITERAL
CLOSURE-ANNOTATE-MV-CALL CLOSURE-ANNOTATE-MV-PROG1 CLOSURE-ANNOTATE-OPCODES
CLOSURE-ANNOTATE-PROGN CLOSURE-ANNOTATE-PROGV CLOSURE-ANNOTATE-RETURN CLOSURE-ANNOTATE-SETQ
CLOSURE-ANNOTATE-TAGBODY CLOSURE-ANNOTATE-THROW CLOSURE-ANNOTATE-UNWIND-PROTECT
CLOSURE-ANNOTATE-VAR-REF)

;; Testing annotation

(IL:FUNCTIONS TEST-ANNOTATION)

;; Arrange to use the proper compiler.

(IL:PROP IL:FILETYPE IL:XCLC-ANNOTATE)

;; Arrange for the proper makefile-environment

(IL:PROP IL:MAKEFILE-ENVIRONMENT IL:XCLC-ANNOTATE)))

;; Annotation of the program tree

(DEFUN **ANNOTATE-TREE** (TREE)

;; Perform those analyses of the program that are not used by meta-evaluation and which may be expensive to redo.

(**FRAME-ANNOTATE** TREE)

(**CLOSURE-ANNOTATE** TREE NIL)
TREE)

;; Frame Annotation

(DEFVAR ***REFERENCES*** NIL

"A list of pairs <go-or-return . tagbody-or-block> representing references to blippers below this point.
Used in frame analysis.")

(DEFUN **FRAME-ANNOTATE** (NODE)

;;; Frame annotation methods are used to discover which constructs in the program need separate frames at run time. The limitations are that some
;;; LAMBDA's are too complex to be treated inline and that the various blippers (TAGBODY, BLOCK, and CATCH) must share the blip slot among
;;; themselves. No two blippers can be using the slot at the same time, so the inner of the two must use a new frame and, thus, a new blip slot. Both
;;; TAGBODY and BLOCK require the blip slot if and only if they can be dynamically separated from a reference to them (i.e., a corresponding GO or
;;; RETURN-FROM). We call this being "closed-over".

;;; Thus, frame analysis works as follows:

;;; Referencers (GO and RETURN-FROM) push a pair <referrer . referree> onto the special variable *REFERENCES*.

;;; Constructs that can require new frames (blippers and LAMBDA's) bind *REFERENCES* so that they can mark as closed over the referrees that will
;;; be separated from their referencers.

;;; Possible referrees (TAGBODY and BLOCK) bind *REFERENCES* so that they can filter out any references to themselves before letting the list go
;;; higher in the tree.

;;; Blippers "request" use of the blip slot by returning themselves from the method. All nodes keep track of the list of requests from below them and
;;; return that list. Such lists may be destructively NCONC'ed. An invariant of the request list is that all of the requests are mutually exclusive; i.e., no
;;; two are nested, one within the other.

;; To maintain this invariant, any blipper that makes a request must tell each of its subordinate requestors that they must be separate frames. Those requestors, in turn, must arrange that the references below them are made aware of their new closed-over status.

;; The astute reader who notices the possibility of infinite regress at this point is to be reassured. The full explanation, in the XCL Compiler Implementor's Handbook, contains a proof of the algorithm's correctness.

```
(AND NODE (NODE-DISPATCH FRAME-ANNOTATE NODE)) )
```

```
(DEFUN CLOSE-OVER (REFERENCE)
  (ETYPESCASE (CAR REFERENCE)
    (GO-NODE (LET ((TAGBODY (CDR REFERENCE))
                  (TAG (GO-TAG (CAR REFERENCE))))
              (SETF (TAGBODY-CLOSED-OVER-P TAGBODY)
                    T)
              (SETF (SEGMENT-CLOSED-OVER-P (FIND-SEGMENT TAGBODY TAG)
                    T)))
    (RETURN-NODE (SETF (BLOCK-CLOSED-OVER-P (CDR REFERENCE))
                      T))))
```

```
(DEFUN FRAME-ANNOTATE-BLOCK (SELF)
```

;; This is one of the interesting ones. If we are closed over, tell the requests that they need to be separate frames, tell their lists of references that they're closed-over, save our references, and request a blip for us. If we're not closed over, then become a requestor only if we received some requests.

```
(LET (OUTER-REFERENCES REQUESTS)
  (LET (*REFERENCES*)
    (SETQ REQUESTS (FRAME-ANNOTATE (BLOCK-STMT SELF)))
    (SETQ OUTER-REFERENCES (DELETE-IF #'(LAMBDA (REFERENCE)
                                          (EQ SELF (CDR REFERENCE)))
                                      *REFERENCES*))
    (COND
      ((BLOCK-CLOSED-OVER-P SELF)
       (IL:FOR REQUESTOR IL:IN REQUESTS IL:DO (SETF (BLIPPER-NEW-FRAME-P REQUESTOR)
                                                    T)
        (IL:FOR REFERENCE IL:IN (BLIPPER-REFERENCES REQUESTOR)
          IL:DO (CLOSE-OVER REFERENCE)))
       (SETF (BLIPPER-REFERENCES SELF)
             (COPY-LIST OUTER-REFERENCES))
       (SETQ REQUESTS (LIST SELF))
       ((NOT (NULL REQUESTS))
        (SETQ REQUESTS (LIST SELF))))
    (SETQ *REFERENCES* (NUNION OUTER-REFERENCES *REFERENCES* :TEST 'EQUAL))
    REQUESTS))
```

```
(DEFUN FRAME-ANNOTATE-CALL (SELF)
```

;; Check for a LAMBDA in the function position and let it know that's where it is.

```
(NCONC (IF (LAMBDA-P (CALL-FN SELF))
          (FRAME-ANNOTATE-LAMBDA (CALL-FN SELF)
           T)
        (FRAME-ANNOTATE (CALL-FN SELF)))
  (IL:FOR ARG IL:IN (CALL-ARGS SELF) IL:JOIN (FRAME-ANNOTATE ARG)))
```

```
(DEFUN FRAME-ANNOTATE-CATCH (SELF)
```

;; This is one of the interesting ones. A CATCH always needs a blip slot, but it can share that slot with other blippers.

```
(LET (OUTER-REFERENCES)
  (LET (*REFERENCES*)
    (SETQ OUTER-REFERENCES *REFERENCES*)
    (IL:FOR REQUESTOR IL:IN (FRAME-ANNOTATE (CATCH-STMT SELF)) IL:DO (SETF (BLIPPER-NEW-FRAME-P
                                                                                   REQUESTOR)
                                                                              T)
      (IL:FOR REFERENCE
        IL:IN (BLIPPER-REFERENCES REQUESTOR)
        IL:DO (CLOSE-OVER REFERENCE)))
    (SETQ OUTER-REFERENCES *REFERENCES*)
    (SETF (BLIPPER-REFERENCES SELF)
          (COPY-LIST *REFERENCES*))
    (SETQ *REFERENCES* (NUNION OUTER-REFERENCES *REFERENCES* :TEST 'EQUAL))
    (CONS SELF (FRAME-ANNOTATE (CATCH-TAG SELF)))))
```

```
(DEFUN FRAME-ANNOTATE-GO (SELF)
```

;; This is one of the interesting ones. Add this GO to the list of references.

```
(PUSHNEW (CONS SELF (GO-TAGBODY SELF))
  *REFERENCES* :TEST 'EQUAL)
NIL)
```

```
(DEFUN FRAME-ANNOTATE-IF (SELF)
```

```
(NCONC (FRAME-ANNOTATE (IF-PRED SELF))
      (FRAME-ANNOTATE (IF-THEN SELF))
      (FRAME-ANNOTATE (IF-ELSE SELF))))
```

```
(DEFUN FRAME-ANNOTATE-LABELS (SELF)
  (NCONC (IL:FOR PAIR IL:IN (LABELS-FUNS SELF) IL:JOIN (FRAME-ANNOTATE (CDR PAIR)))
        (FRAME-ANNOTATE (LABELS-BODY SELF))))
```

```
(DEFUN FRAME-ANNOTATE-LAMBDA (SELF &OPTIONAL FUNCTIONAL-POSITION-P)
```

```
;; This is one of the interesting ones.
```

```
;; The first thing is to decide whether or not this lambda must be a separate frame. If it has only required parameters and appears in functional position,
;; then it need not be a separate frame. However, Interlisp's lambda no-spread's cannot be separate frames.
```

```
;; If this lambda is not a separate frame, then it transparently passes along the requests from below. Otherwise, however, it will be a separate frame
;; and can accept the requests from below. It thus discards the requests and posts a new, empty list. Also, any references from below are closed over.
```

```
(SETF (LAMBDA-NEW-FRAME-P SELF)
      (OR (NOT FUNCTIONAL-POSITION-P)
          (LAMBDA-KEYWORD SELF)
          (LAMBDA-REST SELF)
          (LAMBDA-OPTIONAL SELF)
          (EQ 2 (LAMBDA-ARG-TYPE SELF))))
(LET (OUTER-REFERENCES OUTER-REQUESTS)
  (LET (*REFERENCES*)
    (LET ((REQUESTS (NCONC (IL:FOR OPT-VAR IL:IN (LAMBDA-OPTIONAL SELF) IL:JOIN (FRAME-ANNOTATE
                                                                                     (SECOND OPT-VAR)))
                          (IL:FOR KEY-VAR IL:IN (LAMBDA-KEYWORD SELF) IL:JOIN (FRAME-ANNOTATE
                                                                                     (THIRD KEY-VAR)))
                          (FRAME-ANNOTATE (LAMBDA-BODY SELF))))))
      (COND
        ((LAMBDA-NEW-FRAME-P SELF) ; We're opaque.
         (SETQ OUTER-REQUESTS NIL)
         (SETQ OUTER-REFERENCES NIL)
         (IL:FOR REFERENCE IL:IN *REFERENCES* IL:DO (CLOSE-OVER REFERENCE)))
        (T ; We're transparent.
         (SETQ OUTER-REQUESTS REQUESTS)
         (SETQ OUTER-REFERENCES *REFERENCES*))))
  (SETQ *REFERENCES* (NUNION OUTER-REFERENCES *REFERENCES* :TEST 'EQUAL))
  OUTER-REQUESTS))
```

```
(DEFUN FRAME-ANNOTATE-LITERAL (SELF)
  NIL)
```

```
(DEFUN FRAME-ANNOTATE-MV-CALL (SELF)
  (NCONC (FRAME-ANNOTATE (MV-CALL-FN SELF))
        (IL:FOR ARG IL:IN (MV-CALL-ARG-EXPRS SELF) IL:JOIN (FRAME-ANNOTATE ARG))))
```

```
(DEFUN FRAME-ANNOTATE-MV-PROG1 (SELF)
  (IL:for| STMT IL:|in| (MV-PROG1-STMTS SELF) IL:|join| (FRAME-ANNOTATE STMT)))
```

```
(DEFUN FRAME-ANNOTATE-OPCODES (SELF)
  NIL)
```

```
(DEFUN FRAME-ANNOTATE-PROGN (SELF)
  (IL:FOR STMT IL:IN (PROGN-STMTS SELF) IL:JOIN (FRAME-ANNOTATE STMT)))
```

```
(DEFUN FRAME-ANNOTATE-PROGV (SELF)
  (NCONC (FRAME-ANNOTATE (PROGV-SYMS-EXPR SELF))
        (FRAME-ANNOTATE (PROGV-VALS-EXPR SELF))
        (FRAME-ANNOTATE (PROGV-STMT SELF))))
```

```
(DEFUN FRAME-ANNOTATE-RETURN (SELF)
  ;; This is one of the interesting ones. Add this RETURN to the list of references.
  (PUSHNEW (CONS SELF (RETURN-BLOCK SELF))
    *REFERENCES* :TEST 'EQUAL)
  (FRAME-ANNOTATE (RETURN-VALUE SELF)))
```

```
(DEFUN FRAME-ANNOTATE-SETQ (SELF)
  (FRAME-ANNOTATE (SETQ-VALUE SELF)))
```

```
(DEFUN FRAME-ANNOTATE-TAGBODY (SELF)
```

```
;; This is one of the interesting ones. If we are closed over, tell the requests that they need to be separate frames, tell their lists of references that
;; they're closed-over, save our references, and request a blip for us. If we're not closed over, then become a requestor only if we received some
```

[illegible]

```

      (T (IF (BLOCK-NEW-FRAME-P NODE)
              (SETF (BLOCK-CLOSED-OVER-VARS NODE)
                    *NEED-STORAGE*)
              (SETQ OUTER-NEED-STORAGE *NEED-STORAGE*)))
      (SETQ *NEED-STORAGE* (UNION OUTER-NEED-STORAGE *NEED-STORAGE*)
        NON-LOCALS))

```

```

(DEFUN CLOSURE-ANNOTATE-CALL (NODE IN-LOOP-P)
  (UNION (CLOSURE-ANNOTATE (CALL-FN NODE)
    IN-LOOP-P)
    (IL:FOR ARG IL:IN (CALL-ARGS NODE) UNIONING (CLOSURE-ANNOTATE ARG IN-LOOP-P))))

```

```

(DEFUN CLOSURE-ANNOTATE-CATCH (NODE IN-LOOP-P)

```

;; If we are supposed to be a new frame, then we have to close over all of the non-local references in the catch-stmt and take responsibility for allocating storage for all of the variables in *NEED-STORAGE*. If we aren't a new frame, then we needn't do anything special here.

```

(COND
  ((BLIPPER-NEW-FRAME-P NODE)
   (LET (*NEED-STORAGE*)
     (IL:FOR VAR IL:IN (CLOSURE-ANNOTATE (CATCH-STMT NODE)
      NIL)
      IL:DO (SETF (VARIABLE-CLOSED-OVER VAR)
        T))
     (SETF (CATCH-CLOSED-OVER-VARS NODE)
       *NEED-STORAGE*))
   (CLOSURE-ANNOTATE (CATCH-TAG NODE)
     IN-LOOP-P))
  (T (UNION (CLOSURE-ANNOTATE (CATCH-STMT NODE)
    IN-LOOP-P)
    (CLOSURE-ANNOTATE (CATCH-TAG NODE)
      IN-LOOP-P))))

```

```

(DEFUN CLOSURE-ANNOTATE-GO (NODE IN-LOOP-P)
  NIL)

```

```

(DEFUN CLOSURE-ANNOTATE-IF (NODE IN-LOOP-P)
  (UNION (CLOSURE-ANNOTATE (IF-PRED NODE)
    IN-LOOP-P)
    (UNION (CLOSURE-ANNOTATE (IF-THEN NODE)
      IN-LOOP-P)
      (CLOSURE-ANNOTATE (IF-ELSE NODE)
        IN-LOOP-P))))

```

```

(DEFUN CLOSURE-ANNOTATE-LABELS (NODE IN-LOOP-P)
  (LET (OUTER-NEED-STORAGE NON-LOCALS)
    (LET (*NEED-STORAGE*)
      (SETQ NON-LOCALS (DELETE-IF #'(LAMBDA (VAR)
        (EQ NODE (VARIABLE-BINDER VAR)))
        (UNION (CLOSURE-ANNOTATE (LABELS-BODY NODE)
          NIL)
          (IL:FOR FUN IL:IN (LABELS-FUNS NODE) UNIONING (CLOSURE-ANNOTATE
            (CDR FUN)
            NIL))))))
      (IF (NOT IN-LOOP-P)
        (SETQ OUTER-NEED-STORAGE *NEED-STORAGE*)
        (SETF (LABELS-CLOSED-OVER-VARS NODE)
          *NEED-STORAGE*))
      ;; Now we're outside the scope of the binding of *NEED-STORAGE*
      (SETQ *NEED-STORAGE* (UNION *NEED-STORAGE* OUTER-NEED-STORAGE)
        NON-LOCALS))

```

```

(DEFUN CLOSURE-ANNOTATE-LAMBDA (NODE IN-LOOP-P)
  (LET* ((NEW-FRAME-P (LAMBDA-NEW-FRAME-P NODE))
    OUTER-NEED-STORAGE NON-LOCALS)
    (LET (*NEED-STORAGE*)
      (SETQ NON-LOCALS (UNION (IL:FOR OPT-VAR IL:IN (LAMBDA-OPTIONAL NODE)
        UNIONING (CLOSURE-ANNOTATE (SECOND OPT-VAR)
          NIL))
        (UNION (IL:FOR KEY-VAR IL:IN (LAMBDA-KEYWORD NODE)
          UNIONING (CLOSURE-ANNOTATE (THIRD KEY-VAR)
            NIL))
          (CLOSURE-ANNOTATE (LAMBDA-BODY NODE)
            NIL))))))
      (SETQ NON-LOCALS (DELETE-IF #'(LAMBDA (VAR)
        (EQ (VARIABLE-BINDER VAR)
          NODE))
        NON-LOCALS))
      (WHEN NEW-FRAME-P
        (IL:FOR VAR IL:IN NON-LOCALS IL:DO (SETF (VARIABLE-CLOSED-OVER VAR)

```

```

T))
  (SETQ NON-LOCALS NIL))
  (SETQ *NEED-STORAGE* (APPEND (IL:FOR VAR IL:IN (LAMBDA-REQUIRED NODE) IL:WHEN (VARIABLE-CLOSED-OVER
    VAR)
    IL:COLLECT VAR)
    (IL:FOR OPT-VAR IL:IN (LAMBDA-OPTIONAL NODE)
      IL:JOIN (NCONC (AND (VARIABLE-CLOSED-OVER (FIRST OPT-VAR))
        (LIST (FIRST OPT-VAR)))
        (AND (THIRD OPT-VAR)
          (VARIABLE-CLOSED-OVER (THIRD OPT-VAR))
          (LIST (THIRD OPT-VAR))))))
      (AND (LAMBDA-REST NODE)
        (VARIABLE-CLOSED-OVER (LAMBDA-REST NODE))
        (LIST (LAMBDA-REST NODE))))
    (IL:FOR OPT-VAR IL:IN (LAMBDA-KEYWORD NODE)
      IL:JOIN (NCONC (AND (VARIABLE-CLOSED-OVER (SECOND OPT-VAR))
        (LIST (SECOND OPT-VAR)))
        (AND (FOURTH OPT-VAR)
          (VARIABLE-CLOSED-OVER (FOURTH OPT-VAR))
          (LIST (FOURTH OPT-VAR))))))
    *NEED-STORAGE*))
  (IF (OR NEW-FRAME-P IN-LOOP-P)
    (SETF (LAMBDA-CLOSED-OVER-VARS NODE)
      *NEED-STORAGE*))
  (SETQ OUTER-NEED-STORAGE *NEED-STORAGE*))
  (SETQ *NEED-STORAGE* (UNION OUTER-NEED-STORAGE *NEED-STORAGE*))
  NON-LOCALS))

(DEFUN CLOSURE-ANNOTATE-LITERAL (NODE IN-LOOP-P)
  NIL)

(DEFUN CLOSURE-ANNOTATE-MV-CALL (NODE IN-LOOP-P)
  (UNION (CLOSURE-ANNOTATE (MV-CALL-FN NODE)
    IN-LOOP-P)
    (IL:FOR ARG IL:IN (MV-CALL-ARG-EXPRS NODE) UNIONING (CLOSURE-ANNOTATE ARG IN-LOOP-P))))

(DEFUN CLOSURE-ANNOTATE-MV-PROG1 (NODE IN-LOOP-P)
  (IL:FOR STMT IL:IN (MV-PROG1-STMTS NODE) UNIONING (CLOSURE-ANNOTATE STMT IN-LOOP-P)))

(DEFUN CLOSURE-ANNOTATE-OPCODES (NODE IN-LOOP-P)
  NIL)

(DEFUN CLOSURE-ANNOTATE-PROGN (NODE IN-LOOP-P)
  (IL:FOR STMT IL:IN (PROGN-STMTS NODE) UNIONING (CLOSURE-ANNOTATE STMT IN-LOOP-P)))

(DEFUN CLOSURE-ANNOTATE-PROGV (NODE IN-LOOP-P)
  (UNION (CLOSURE-ANNOTATE (PROGV-SYMS-EXPR NODE)
    IN-LOOP-P)
    (UNION (CLOSURE-ANNOTATE (PROGV-VALS-EXPR NODE)
      IN-LOOP-P)
      (CLOSURE-ANNOTATE (PROGV-STMT NODE)
        IN-LOOP-P))))

(DEFUN CLOSURE-ANNOTATE-RETURN (NODE IN-LOOP-P)
  (CLOSURE-ANNOTATE (RETURN-VALUE NODE)
    IN-LOOP-P))

(DEFUN CLOSURE-ANNOTATE-SETQ (NODE IN-LOOP-P)
  (ADJOIN (SETQ-VAR NODE)
    (CLOSURE-ANNOTATE (SETQ-VALUE NODE)
      IN-LOOP-P)))

(DEFUN CLOSURE-ANNOTATE-TAGBODY (NODE IN-LOOP-P)
  (LET (OUTER-NEED-STORAGE NON-LOCALS)
    (LET* (*NEED-STORAGE* (SEGMENTS (TAGBODY-SEGMENTS NODE))
      (COULD-BE-A-LOOP (OR IN-LOOP-P (NOT (NULL (SEGMENT-TAGS (CAR SEGMENTS)))))
        (NOT (NULL (CDR SEGMENTS))))))
      (SETQ NON-LOCALS (IL:FOR SEGMENT IL:IN SEGMENTS UNIONING (IL:FOR STMT IL:IN (SEGMENT-STMTS SEGMENT)
        UNIONING (CLOSURE-ANNOTATE STMT
          COULD-BE-A-LOOP))))))
    (WHEN (TAGBODY-NEW-FRAME-P NODE)
      (IL:FOR VAR IL:IN NON-LOCALS IL:DO (SETF (VARIABLE-CLOSED-OVER VAR)
        T))
      (SETQ NON-LOCALS NIL))
    (COND
      ((TAGBODY-CLOSED-OVER-P NODE)
        (SETF (TAGBODY-BLIP-VAR NODE)
          (MAKE-VARIABLE :SCOPE :LEXICAL :KIND :VARIABLE :NAME "Control blip" :BINDER NODE

```

```

      :CLOSED-OVER T))
    (IF (OR (TAGBODY-NEW-FRAME-P NODE)
            IN-LOOP-P)
        (SETF (TAGBODY-CLOSED-OVER-VARS NODE)
              (CONS (TAGBODY-BLIP-VAR NODE)
                    *NEED-STORAGE*))
        (SETQ OUTER-NEED-STORAGE (CONS (TAGBODY-BLIP-VAR NODE)
                                         *NEED-STORAGE*)))
    (T (IF (TAGBODY-NEW-FRAME-P NODE)
          (SETF (TAGBODY-CLOSED-OVER-VARS NODE)
                *NEED-STORAGE*)
          (SETQ OUTER-NEED-STORAGE *NEED-STORAGE*))))
    (SETQ *NEED-STORAGE* (UNION OUTER-NEED-STORAGE *NEED-STORAGE*))
    NON-LOCALS))

```

```

(DEFUN CLOSURE-ANNOTATE-THROW (NODE IN-LOOP-P)
  (UNION (CLOSURE-ANNOTATE (THROW-TAG NODE)
                            IN-LOOP-P)
        (CLOSURE-ANNOTATE (THROW-VALUE NODE)
                            IN-LOOP-P)))

```

```

(DEFUN CLOSURE-ANNOTATE-UNWIND-PROTECT (NODE IN-LOOP-P)
  (UNION (CLOSURE-ANNOTATE (UNWIND-PROTECT-STMT NODE)
                            IN-LOOP-P)
        (CLOSURE-ANNOTATE (UNWIND-PROTECT-CLEANUP NODE)
                            IN-LOOP-P)))

```

```

(DEFUN CLOSURE-ANNOTATE-VAR-REF (NODE IN-LOOP-P)
  (LET ((VAR (VAR-REF-VARIABLE NODE)))
    (IF (EQ (VARIABLE-SCOPE VAR)
            :LEXICAL)
        (LIST VAR))))

```

:: Testing annotation

```

(DEFUN TEST-ANNOTATION (FN)
  (LET ((TREE (TEST-ALPHA-2 FN)))
    (UNWIND-PROTECT
     (PRINT-TREE (ANNOTATE-TREE (META-EVALUATE TREE)))
     (RELEASE-TREE TREE))))

```

:: Arrange to use the proper compiler.

```
(IL:PUTPROPS IL:XCLC-ANNOTATE IL:FILETYPE COMPILE-FILE)
```

:: Arrange for the proper makefile-environment

```
(IL:PUTPROPS IL:XCLC-ANNOTATE IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE (DEFPACKAGE "COMPILER"
                                                (:USE "LISP" "XCL"))))
```

```
(IL:PUTPROPS IL:XCLC-ANNOTATE IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1988 1990))
```

FUNCTION INDEX

ANNOTATE-TREE	1	CLOSURE-ANNOTATE-PROGV	6	FRAME-ANNOTATE-LITERAL	3
CLOSE-OVER	2	CLOSURE-ANNOTATE-RETURN	6	FRAME-ANNOTATE-MV-CALL	3
CLOSURE-ANNOTATE	4	CLOSURE-ANNOTATE-SETQ	6	FRAME-ANNOTATE-MV-PROG1	3
CLOSURE-ANNOTATE-BLOCK	4	CLOSURE-ANNOTATE-TAGBODY	6	FRAME-ANNOTATE-OPCODES	3
CLOSURE-ANNOTATE-CALL	5	CLOSURE-ANNOTATE-THROW	7	FRAME-ANNOTATE-PROGN	3
CLOSURE-ANNOTATE-CATCH	5	CLOSURE-ANNOTATE-UNWIND-PROTECT ..	7	FRAME-ANNOTATE-PROGV	3
CLOSURE-ANNOTATE-GO	5	CLOSURE-ANNOTATE-VAR-REF	7	FRAME-ANNOTATE-RETURN	3
CLOSURE-ANNOTATE-IF	5	FRAME-ANNOTATE	1	FRAME-ANNOTATE-SETQ	3
CLOSURE-ANNOTATE-LABELS	5	FRAME-ANNOTATE-BLOCK	2	FRAME-ANNOTATE-TAGBODY	3
CLOSURE-ANNOTATE-LAMBDA	5	FRAME-ANNOTATE-CALL	2	FRAME-ANNOTATE-THROW	4
CLOSURE-ANNOTATE-LITERAL	6	FRAME-ANNOTATE-CATCH	2	FRAME-ANNOTATE-UNWIND-PROTECT	4
CLOSURE-ANNOTATE-MV-CALL	6	FRAME-ANNOTATE-GO	2	FRAME-ANNOTATE-VAR-REF	4
CLOSURE-ANNOTATE-MV-PROG1	6	FRAME-ANNOTATE-IF	2	TEST-ANNOTATION	7
CLOSURE-ANNOTATE-OPCODES	6	FRAME-ANNOTATE-LABELS	3		
CLOSURE-ANNOTATE-PROGN	6	FRAME-ANNOTATE-LAMBDA	3		

VARIABLE INDEX

NEED-STORAGE	4	*REFERENCES*	1
----------------------	---	--------------------	---

PROPERTY INDEX

IL:XCLC-ANNOTATE	7
------------------------	---

I.S.OPR INDEX

UNIONING	4
----------------	---
