File created: 10-Apr-2024 19:21:49 **{DSK}<home>larry>il>medley>sources>XCLC-ALPHA.;2**

edit by: lmm

changes to: (IL:FUNCTIONS ALPHA-COMPILER-LET ALPHA-FLET ALPHA-LAMBDA ALPHA-LET ALPHA-LET* ALPHA-MACROLET ALPHA-PROGN ALPHA-SETQ ALPHA-TAGBODY COMPLETELY-EXPAND EXPAND-OPENLAMBDA-CALL PRINT-NODE)

previous date: 21-Mar-2024 10:27:05 {DSK}<home>larry>il>medley>sources>XCLC-ALPHA.;1

Read Table: XCL

Package: COMPILER

Format: XCCS

```
(IL:RPAQQ IL:XCLC-ALPHACOMS
          (

;;; Alphatization

           (IL:FUNCTIONS BINDING-CONTOUR PROCESS-DECLARATIONS PROCESS-IL-DECLARATIONS UPDATE-ENVIRONMENT)
           (IL:FUNCTIONS BIND-PARAMETER CHECK-ARG)
           (IL:FUNCTIONS BINDING-TO-LAMBDA)
           (IL:VARIABLES *BLOCK-STACK* *TAGBODY-STACK*)
           (IL:FUNCTIONS ALPHA-ARGUMENT-FORM ALPHA-ATOM ALPHA-BLOCK ALPHA-CATCH ALPHA-COMBINATION
                  ALPHA-COMPILER-LET ALPHA-EVAL-WHEN ALPHA-FLET ALPHA-FORM ALPHA-FUNCTION ALPHA-FUNCTIONAL-FORM
                  ALPHA-GO ALPHA-IF ALPHA-IL-FUNCTION ALPHA-LABELS ALPHA-LAMBDA ALPHA-LAMBDA-LIST ALPHA-LET
                  ALPHA-LET* ALPHA-LITERAL ALPHA-MACROLET ALPHA-MV-CALL ALPHA-MV-PROG1 ALPHA-PROGN ALPHA-PROGV
                  ALPHA-RETURN-FROM ALPHA-SETQ ALPHA-TAGBODY ALPHA-THROW ALPHA-UNWIND-PROTECT)
           (IL:FUNCTIONS CONVERT-TO-CL-LAMBDA COMPLETELY-EXPAND EXPAND-OPENLAMBDA-CALL)

           ;; Alphatization testing

           (IL:VARIABLES *INDENT-INCREMENT* *NODE-HASH* *NODE-NUMBER*)
           (IL:FUNCTIONS TEST-ALPHA TEST-ALPHA-2 PARSE-DEFUN PRINT-TREE PRINT-NODE)
           (IL:VARIABLES CONTEXT-TEST-FORM)
           (IL:FUNCTIONS CTXT)

           ;; Arrange to use the correct compiler.

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

           ;; Arrange for the correct makefile environment

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

;;; Alphatization

(DEFMACRO BINDING-CONTOUR (DECLARATIONS &BODY BODY)

;;; Called around the alphatization of a binding form, this sets up bindings of the various special variables used to communicate information between
;;; declarations and code.  The given declarations are then processed inside the bindings before going on to the body.

    `(LET ((*NEW-SPECIALS* NIL)
           (*NEW-GLOBALS* NIL)
           (*NEW-INLINES* NIL)
           (*NEW-NOTINLINES* NIL)
           (IL:SPECVARS IL:SPECVARS)
           (IL:LOCALVARS IL:LOCALVARS)
           (IL:GLOBALVARS IL:GLOBALVARS))
          (DECLARE (SPECIAL *NEW-SPECIALS* *NEW-GLOBALS* *NEW-INLINES* *NEW-NOTINLINES* IL:SPECVARS IL:LOCALVARS
                            IL:GLOBALVARS))
          (PROCESS-DECLARATIONS ,DECLARATIONS)
          ,@BODY))

(DEFUN PROCESS-DECLARATIONS (DECLS)                    ; Edited 21-Mar-2024 10:26 by lmm

;;; Step through the given declarations, storing the information found therein into various special variables.

    (DECLARE (SPECIAL *NEW-SPECIALS* *NEW-GLOBALS* *NEW-INLINES* *NEW-NOTINLINES* IL:SPECVARS IL:LOCALVARS
                      IL:GLOBALVARS))
    (FLET ((CHECK-VAR-1 (VAR)
                (COND
                    ((SYMBOLP VAR)
                     VAR)
                    (T (CERROR "Use the symbol %LOSE% instead." "The value ~S, appearing in a declaration, is
                               not a symbol" VAR)
                       '%LOSE%))))
           (MACROLET ((CHECK-VAR (VAR)
                         `(SETQ ,VAR (CHECK-VAR-1 ,VAR))))
                  (DOLIST (DECL DECLS)
                      (DOLIST (SPEC (CDR DECL))
                          (IF (ATOM SPEC)
                              (CERROR "Ignore it." "A non-list, ~S, was found where a declaration specification
                                     was expected." SPEC)
                              (CASE (CAR SPEC)
                                  ((SPECIAL) (DOLIST (VAR (CDR SPEC))
                                                 (CHECK-VAR VAR)
```

```
                                (PUSH VAR *NEW-SPECIALS*)))
                        ((IL:SPECVARS) (COND
                                          ((CONSP (CDR SPEC))
                                           (UNLESS (EQ IL:SPECVARS T)
                                               (SETQ IL:SPECVARS (UNION IL:SPECVARS (CDR SPEC)))))
                                          ((EQ (CDR SPEC)
                                               T)
                                           (SETQ IL:SPECVARS T)
                                           (SETQ IL:LOCALVARS IL:SYSLOCALVARS))
                                          (T (CERROR "Ignore it" "Illegal SPECVARS declaration: ~S" SPEC
                                                     ))))
                        ((IL:LOCALVARS) (COND
                                           ((CONSP (CDR SPEC))
                                            (UNLESS (EQ IL:LOCALVARS T)
                                                (SETQ IL:LOCALVARS (UNION IL:LOCALVARS (CDR SPEC)))))
                                           ((EQ (CDR SPEC)
                                                T)
                                            (SETQ IL:LOCALVARS T)
                                            (SETQ IL:SPECVARS IL:SYSSPECVARS))
                                           (T (CERROR "Ignore it" "Illegal LOCALVARS declaration: ~S"
                                                      SPEC))))
                        ((GLOBAL) (DOLIST (VAR (CDR SPEC))
                                    (CHECK-VAR VAR)
                                    (PUSH VAR *NEW-GLOBALS*)))
                        ((IL:GLOBALVARS) (IF (CONSP (CDR SPEC))
                                             (SETQ IL:GLOBALVARS (UNION IL:GLOBALVARS (CDR SPEC)))
                                             (CERROR "Ignore it" "Illegal GLOBALVARS declaration: ~S"
                                                     SPEC)))
                        ((TYPE FTYPE FUNCTION)           ; We don't handle type declarations yet.
                           NIL)
                        ((INLINE) (DOLIST (VAR (CDR SPEC))
                                    (CHECK-VAR VAR)
                                    (PUSH VAR *NEW-INLINES*)))
                        ((NOTINLINE) (DOLIST (VAR (CDR SPEC))
                                       (CHECK-VAR VAR)
                                       (PUSH VAR *NEW-NOTINLINES*)))
                        ((IGNORE OPTIMIZE IGNORABLE)     ; We don't handle IGNORE or OPTIMIZE declarations yet.
                           NIL)
                        ((DECLARATION)                   ; Add new declaration specifiers right away so that they can be
                                                         ; used in later declarations in the same cluster.  It's a picky point,
                                                         ; but who cares?
                           (ENV-ADD-DECLS *ENVIRONMENT* (CDR SPEC)))
                        ((IL:USEDFREE)                   ; Ignored Interlisp declarations
                           NIL)
                        (OTHERWISE (UNLESS (OR (EQ (CAR SPEC)
                                                   T)
                                               (IL:TYPE-EXPANDER (CAR SPEC))
                                               (XCL::DECL-SPECIFIER-P (CAR SPEC))
                                               (ENV-DECL-P *ENVIRONMENT* (CAR SPEC)))
                                           (CERROR "Ignore it." "Unknown declaration specifier in DECLARE:
                                                  ~S." (CAR SPEC))))))))))
```

(DEFUN **PROCESS-IL-DECLARATIONS** (SPECS)

;;; Stroring theInterlisp's declare information found in executable position.

```
   (DECLARE (SPECIAL IL:SPECVARS IL:LOCALVARS IL:GLOBALVARS))
   (DOLIST (SPEC SPECS T)
       (IF (ATOM SPEC)
           (CERROR "Ignore it." "A non-list, ~S, was found where a declaration specification was expected." SPEC
                  )
           (CASE (CAR SPEC)
               ((IL:SPECVARS) (COND
                                 ((CONSP (CDR SPEC))
                                  (UNLESS (EQ IL:SPECVARS T)
                                      (SETQ IL:SPECVARS (UNION IL:SPECVARS (CDR SPEC)))))
                                 ((EQ (CDR SPEC)
                                      T)
                                  (SETQ IL:SPECVARS T)
                                  (SETQ IL:LOCALVARS IL:SYSLOCALVARS))
                                 (T (CERROR "Ignore it" "Illegal SPECVARS declaration: ~S" SPEC))))
               ((IL:LOCALVARS) (COND
                                  ((CONSP (CDR SPEC))
                                   (UNLESS (EQ IL:LOCALVARS T)
                                       (SETQ IL:LOCALVARS (UNION IL:LOCALVARS (CDR SPEC)))))
                                  ((EQ (CDR SPEC)
                                       T)
                                   (SETQ IL:LOCALVARS T)
                                   (SETQ IL:SPECVARS IL:SYSSPECVARS))
                                  (T (CERROR "Ignore it" "Illegal LOCALVARS declaration: ~S" SPEC))))
               ((IL:GLOBALVARS) (IF (CONSP (CDR SPEC))
                                    (SETQ IL:GLOBALVARS (UNION IL:GLOBALVARS (CDR SPEC)))
                                    (CERROR "Ignore it" "Illegal GLOBALVARS declaration: ~S" SPEC)))
               ((IL:USEDFREE)                            ; Ignored Interlisp declarations
                  NIL)
               (OTHERWISE (RETURN-FROM PROCESS-IL-DECLARATIONS NIL))))))
```

```
(DEFUN UPDATE-ENVIRONMENT (ENV)

;;; Store the information in a BINDING-CONTOUR's special variables into the given environment.

   (DECLARE (SPECIAL *NEW-SPECIALS* *NEW-GLOBALS* *NEW-INLINES* *NEW-NOTINLINES*))
   (WHEN *NEW-SPECIALS* (ENV-DECLARE-SPECIALS ENV *NEW-SPECIALS*))
   (WHEN *NEW-GLOBALS* (ENV-DECLARE-GLOBALS ENV *NEW-GLOBALS*))
   (WHEN *NEW-INLINES* (ENV-ALLOW-INLINES ENV *NEW-INLINES*))
   (WHEN *NEW-NOTINLINES* (ENV-DISALLOW-INLINES ENV *NEW-NOTINLINES*)))


(DEFUN BIND-PARAMETER (VAR BINDER ENV)
   (ECASE (RESOLVE-VARIABLE-BINDING ENV VAR)
       (:SPECIAL
          (DELETEF VAR *NEW-SPECIALS*)
          (ENV-DECLARE-A-SPECIAL ENV VAR)
          (MAKE-VARIABLE :SCOPE :SPECIAL :KIND :VARIABLE :NAME VAR :BINDER BINDER))
       (:LEXICAL (LET ((STRUCT (MAKE-VARIABLE :SCOPE :LEXICAL :KIND :VARIABLE :NAME (SYMBOL-NAME VAR)
                                              :BINDER BINDER)))
                   (ENV-BIND-VARIABLE ENV VAR STRUCT)
                   STRUCT))))

(DEFUN CHECK-ARG (VAR)

;;; Make sure that VAR is a legal parameter in a lambda-list.

   (COND
      ((NOT (SYMBOLP VAR))
       (CERROR "Ignore it." "The parameter ~S is not a symbol." VAR)
       NIL)
      ((KEYWORDP VAR)
       (CERROR "Ignore it." "The parameter ~S is a keyword and may not be bound." VAR)
       NIL)
      (T T)))

(DEFUN BINDING-TO-LAMBDA (BINDING)

;;; Convert a binding from an FLET or LABELS into the appropriate LAMBDA form, wrapping a BLOCK around the bodies of the functions.

   (DESTRUCTURING-BIND (NAME ARG-LIST &BODY BODY)
          BINDING
          (MULTIPLE-VALUE-BIND (FORMS DECLS)
              (PARSE-BODY BODY *ENVIRONMENT* T)
            `(LAMBDA ,ARG-LIST ,@DECLS (BLOCK ,NAME ,@FORMS)))))

(DEFVAR *BLOCK-STACK* NIL

;;; Association list of block names to block structures; rebound at several points within the alphatizer.

   )


(DEFVAR *TAGBODY-STACK* NIL
   "Association list from TAGBODY tags to the TAGBODY structure containing the tag; rebound at several points in
   the alphatizer")

(DEFUN ALPHA-ARGUMENT-FORM (FORM)
   (LET ((*CONTEXT* *ARGUMENT-CONTEXT*))
        (ALPHA-FORM FORM)))

(DEFUN ALPHA-ATOM (FORM)

;;; The form is atomic.  If it's a symbol, do the appropriate look-ups.  Otherwise, it must be a literal.

   (IF (OR (NOT (SYMBOLP FORM))
           (EQ FORM T)
           (EQ FORM NIL))
       (ALPHA-LITERAL FORM)
       (RESOLVE-VARIABLE-REFERENCE *ENVIRONMENT* FORM)))


(DEFUN ALPHA-BLOCK (NAME BODY)
   (LET* ((NEW-BLOCK (MAKE-BLOCK :NAME NAME :CONTEXT *CONTEXT*))
          (*BLOCK-STACK* (CONS (CONS NAME NEW-BLOCK)
                               *BLOCK-STACK*)))
         (SETF (BLOCK-STMT NEW-BLOCK)
               (ALPHA-PROGN BODY))
         NEW-BLOCK))
```

```
(DEFUN ALPHA-CATCH (TAG FORMS)
   (MAKE-CATCH :TAG (ALPHA-ARGUMENT-FORM TAG)
          :STMT
          (ALPHA-PROGN FORMS)))


(DEFUN ALPHA-COMBINATION (FN ARGS)
   (DECLARE (SPECIAL IL:NLAMA IL:NLAML))
   (COND
      ;; Calls to FUNCALL are expanded into CALL nodes where the FN is the first argument to FUNCALL, more or less.
      ((AND (EQ FN 'FUNCALL)
            (NOT (ENV-INLINE-DISALLOWED *ENVIRONMENT* FN)))
       (MULTIPLE-VALUE-BIND (REAL-FN NOT-INLINE?)
           (ALPHA-FUNCTIONAL-FORM (FIRST ARGS))
         (MAKE-CALL :FN REAL-FN :ARGS (MAPCAR #'ALPHA-ARGUMENT-FORM (REST ARGS))
                :NOT-INLINE NOT-INLINE?)))
      ;; Calls on IL:OPENLAMBDA's involve lots of hairy processing.
      ((AND (CONSP FN)
            (EQ (FIRST FN)
                'IL:OPENLAMBDA))
       (ALPHA-FORM (EXPAND-OPENLAMBDA-CALL FN ARGS)))
      ;; Lexical functions and non-symbol functions can't be NLambda's.
      ((OR (NOT (SYMBOLP FN))
           (ENV-FBOUNDP *ENVIRONMENT* FN))
       (MAKE-CALL :FN (ALPHA-FUNCTION FN *CONTEXT*)
              :ARGS
              (MAPCAR #'ALPHA-ARGUMENT-FORM ARGS)
              :NOT-INLINE
              (AND (SYMBOLP FN)
                   (ENV-INLINE-DISALLOWED *ENVIRONMENT* FN))))
      ((OR (EQ 3 (IL:ARGTYPE FN))
           (MEMBER FN IL:NLAMA :TEST 'EQ))                    ; It's an NLambda no-spread.  Funcall it on a single literal
                                                              ; argument, the CDR of the form.
       (MAKE-CALL :FN (ALPHA-FUNCTION FN)
              :ARGS
              (ALPHA-LITERAL ARGS)
              :NOT-INLINE
              (ENV-INLINE-DISALLOWED *ENVIRONMENT* FN)))
      ((OR (EQ 1 (IL:ARGTYPE FN))
           (MEMBER FN IL:NLAML :TEST 'EQ))                    ; It's an NLambda spread.  Funcall it on the quoted versions of its
                                                              ; arguments.
       (MAKE-CALL :FN (ALPHA-FUNCTION FN)
              :ARGS
              (MAPCAR #'ALPHA-LITERAL ARGS)
              :NOT-INLINE
              (ENV-INLINE-DISALLOWED *ENVIRONMENT* FN)))
      (T (MAKE-CALL :FN (ALPHA-FUNCTION FN *CONTEXT*)
                :ARGS
                (MAPCAR #'ALPHA-ARGUMENT-FORM ARGS)
                :NOT-INLINE
                (ENV-INLINE-DISALLOWED *ENVIRONMENT* FN)))))


(DEFUN ALPHA-COMPILER-LET (BINDINGS BODY)
   (LET ((VARS NIL)
         (VALS NIL))
        (IL:FOR BINDING IL:IN BINDINGS IL:DO (COND
                                                ((CONSP BINDING)
                                                 (PUSH (CAR BINDING)
                                                       VARS)
                                                 (PUSH (EVAL (CADR BINDING))
                                                       VALS))
                                                (T (PUSH BINDING VARS)
                                                   (PUSH NIL VALS))))
        (PROGV VARS VALS
            (ALPHA-PROGN BODY))))


(DEFUN ALPHA-EVAL-WHEN (TIMES FORMS)

;;; If the times contain COMPILE, we evaluate the forms.  If the times include LOAD, we prognify the forms.  If LOAD isn't mentioned, this turns into NIL.

   (WHEN (OR (MEMBER 'COMPILE TIMES :TEST #'EQ)
             (MEMBER 'IL:COMPILE TIMES :TEST #'EQ))
       (MAPC #'EVAL FORMS))
   (IF (OR (MEMBER 'LOAD TIMES :TEST #'EQ)
           (MEMBER 'IL:LOAD TIMES :TEST #'EQ))
       (ALPHA-PROGN FORMS)
       *LITERALLY-NIL*))


(DEFUN ALPHA-FLET (BINDINGS BODY)

;;; An FLET is alphatized as a LABELS node. The only difference is that the new variables for the function bindings are inserted after alphatizing the
;;; defined functions and body, whereas in a LABELS you add them to the environment before alphatizing the children.
```

```
   (LET ((*ENVIRONMENT* (MAKE-CHILD-ENV *ENVIRONMENT*)))
        (MULTIPLE-VALUE-BIND (FORMS DECLS)
            (PARSE-BODY BODY *ENVIRONMENT* NIL)
          (BINDING-CONTOUR DECLS (UPDATE-ENVIRONMENT *ENVIRONMENT*)
                 (LET ((NEW-LABELS (MAKE-LABELS))
                       NAMES)
                      (SETQ NAMES (WITH-COLLECTION (SETF (LABELS-FUNS NEW-LABELS)
                                                         (MAPCAR #'(LAMBDA (BINDING)
                                                                     (UNLESS (CHECK-ARG (CAR BINDING))
                                                                         (SETQ BINDING (CONS '%LOSE%
                                                                                             (CDR BINDING))
                                                                               ))
                                                                     (COLLECT (CAR BINDING))
                                                                     (CONS (MAKE-VARIABLE
                                                                            :NAME
                                                                            (SYMBOL-NAME (CAR BINDING))
                                                                            :SCOPE :LEXICAL :KIND :FUNCTION
                                                                            :BINDER NEW-LABELS)
                                                                           (ALPHA-LAMBDA
                                                                            (BINDING-TO-LAMBDA BINDING)
                                                                            :NAME
                                                                            ;; Really want name to be "Foo in Bar"
                                                                            (SYMBOL-NAME (CAR BINDING)))))
                                                                 BINDINGS))))
                      ;; Having alphatized the function bindings, put them in the environment for alphatization of the body.
                      (IL:FOR NAME IL:IN NAMES IL:AS FN-PAIR IL:IN (LABELS-FUNS NEW-LABELS)
                         IL:DO (ENV-BIND-FUNCTION *ENVIRONMENT* NAME :FUNCTION (CAR FN-PAIR)))
                      ;; Now we can alphatize the body.
                      (SETF (LABELS-BODY NEW-LABELS (ALPHA-PROGN FORMS)))
                      NEW-LABELS)))))
```

(DEFUN **ALPHA-FORM** (FORM)

;;; FORM is a random exectuable form. Dispatch to the appropriate alphatization routine.

;;; NOTE NOTE NOTE::: If anything is added to this CASE statement, be sure to add it also to the list in COMPLETELY-EXPAND.

```
   (IF (ATOM FORM)
       (ALPHA-ATOM FORM)
       (CASE (CAR FORM)
           ((BLOCK) (ALPHA-BLOCK (SECOND FORM)
                                 (CDDR FORM)))
           ((CATCH) (ALPHA-CATCH (SECOND FORM)
                                 (CDDR FORM)))
           ((COMPILER-LET) (ALPHA-COMPILER-LET (SECOND FORM)
                                               (CDDR FORM)))
           ((DECLARE)
              (OR (PROCESS-IL-DECLARATIONS (CDR FORM))
                  (CERROR "Replace the declaration with NIL" "DECLARE found in executable position: ~S" FORM))
              *LITERALLY-NIL*)
           ((EVAL-WHEN) (ALPHA-EVAL-WHEN (SECOND FORM)
                                          (CDDR FORM)))
           ((FLET) (ALPHA-FLET (SECOND FORM)
                               (CDDR FORM)))
           ((IL:FUNCTION) (ALPHA-IL-FUNCTION (SECOND FORM)
                                             (THIRD FORM)))
           ((FUNCTION) (ALPHA-FUNCTION (SECOND FORM)))
           ((GO) (ALPHA-GO (SECOND FORM)))
           ((IF) (ALPHA-IF (SECOND FORM)
                           (THIRD FORM)
                           (FOURTH FORM)))
           ((LABELS)                                                ; Rely on the macro expansion for now.
              (RETURN-FROM ALPHA-FORM (ALPHA-LABELS (SECOND FORM)
                                                    (CDDR FORM)))
              (RETURN-FROM ALPHA-FORM (ALPHA-FORM (OPTIMIZE-AND-MACROEXPAND-1 FORM))))
           ((LET) (ALPHA-LET (SECOND FORM)
                             (CDDR FORM)))
           ((LET*) (ALPHA-LET* (SECOND FORM)
                               (CDDR FORM)))
           ((MACROLET SI::%MACROLET) (ALPHA-MACROLET (SECOND FORM)
                                                     (CDDR FORM)))
           ((MULTIPLE-VALUE-CALL) (ALPHA-MV-CALL (SECOND FORM)
                                                 (CDDR FORM)))
           ((MULTIPLE-VALUE-PROG1) (ALPHA-MV-PROG1 (CDR FORM)))
           ((PROGN) (ALPHA-PROGN (CDR FORM)))
           ((PROGV)                                                 ; Rely on the macro expansion for now.
              (RETURN-FROM ALPHA-FORM (DESTRUCTURING-BIND (VARS-EXPR VALS-EXPR . BODY)
                                                          (CDR FORM)
                                                          (ALPHA-FORM `(IL:\\DO.PROGV ,VARS-EXPR ,VALS-EXPR
                                                                                      #'(LAMBDA NIL ,@BODY)))))
              (ALPHA-PROGV (SECOND FORM)
                           (THIRD FORM)
```

```
                    (CDDDR FORM)))
            ((QUOTE) (ALPHA-LITERAL (SECOND FORM)))
            ((RETURN-FROM) (ALPHA-RETURN-FROM (SECOND FORM)
                              (THIRD FORM)))
            ((SETQ IL:SETQ) (ALPHA-SETQ (CAR FORM)
                             (REST FORM)))
            ((TAGBODY) (ALPHA-TAGBODY (CDR FORM)))
            ((THE)                                          ; Ignore the THE construct for now.
               (ALPHA-FORM (THIRD FORM)))
            ((THROW) (ALPHA-THROW (SECOND FORM)
                          (THIRD FORM)))
            ((UNWIND-PROTECT) (ALPHA-UNWIND-PROTECT (SECOND FORM)
                                 (CDDR FORM)))
            (OTHERWISE (MULTIPLE-VALUE-BIND (NEW-FORM CHANGED-P)
                           (OPTIMIZE-AND-MACROEXPAND-1 FORM)
                         (IF (NULL CHANGED-P)
                             (ALPHA-COMBINATION (CAR FORM)
                                    (CDR FORM))
                             (ALPHA-FORM NEW-FORM))))))

(DEFUN ALPHA-FUNCTION (FORM &OPTIONAL (CONTEXT (OR (CONTEXT-APPLIED-CONTEXT *CONTEXT*)
                                                   *NULL-CONTEXT*)))
```

;;; If it's a symbol, then turn this into either the FLET/LABELS-bound VARIABLE structure or a structure for the global symbol. Otherwise, it must be ;;; either a LAMBDA-form or OPCODES-form and is treated as such. Note that the internal representation of programs treats LAMBDA as a ;;; value-producing special form.

;;; The CONTEXT argument is the return-context of the function, if known. It is passed on to alpha-lambda.

;;; We return a second value when the FORM is a symbol, saying whether or not the named function is supposed to be NOTINLINE.

```
   (COND
      ((SYMBOLP FORM)
       (MULTIPLE-VALUE-BIND (KIND STRUCT)
           (ENV-FBOUNDP *ENVIRONMENT* FORM)
         (COND
            ((EQ KIND :FUNCTION)
             (VALUES (MAKE-VAR-REF :VARIABLE STRUCT)
                     (ENV-INLINE-DISALLOWED *ENVIRONMENT* FORM)))
            (T (UNLESS (NULL KIND)
                   (ASSERT (EQ KIND :MACRO))
                   ;; This case can only arise if we are alphatizing a FUNCTION form, since the macro would have been expanded otherwise.
                   (CERROR "Use the global function definition of ~S" "The symbol ~S names a lexically-bound
                          macro and thus cannot be used with the FUNCTION special form." FORM))
               ;; Account for block compilation.
               (WHEN (NOT (NULL *CURRENT-BLOCK*))
                   (LET ((LOOKUP (ASSOC FORM (BLOCK-DECL-FN-NAME-MAP *CURRENT-BLOCK*))))
                        (WHEN (NOT (NULL LOOKUP))                ; This function is to be renamed.
                           (SETQ FORM (CDR LOOKUP)))))
               (CHECK-FOR-UNKNOWN-FUNCTION FORM)
               (VALUES (MAKE-REFERENCE-TO-VARIABLE :NAME FORM :SCOPE :GLOBAL :KIND :FUNCTION)
                       (ENV-INLINE-DISALLOWED *ENVIRONMENT* FORM))))))
      (T (CASE (CAR FORM)
           ((LAMBDA IL:LAMBDA IL:NLAMBDA IL:OPENLAMBDA) (ALPHA-LAMBDA FORM :CONTEXT CONTEXT))
           ((IL:OPCODES :OPCODES) (MAKE-OPCODES :BYTES (CDR FORM)))
           (OTHERWISE
              (CERROR "Use (LAMBDA () NIL) instead" "The form ~S, appearing in a functional context, is
                     neither a symbol nor a LAMBDA-form" FORM)
              (ALPHA-LAMBDA '(LAMBDA NIL NIL)
                     :CONTEXT CONTEXT))))))

(DEFUN ALPHA-FUNCTIONAL-FORM (FORM)
   (IF (AND (CONSP FORM)
            (OR (EQ 'QUOTE (FIRST FORM))
                (EQ 'IL:FUNCTION (FIRST FORM)))
            (SYMBOLP (SECOND FORM)))
       (ALPHA-FUNCTION (SECOND FORM))
       (LET ((*CONTEXT* (MAKE-CONTEXT :VALUES-USED 1 :APPLIED-CONTEXT *CONTEXT*)))
            (ALPHA-FORM FORM))))

(DEFUN ALPHA-GO (TAG)
   (LET ((DEST (ASSOC TAG *TAGBODY-STACK*)))
        (WHEN (NULL DEST)
            (COND
               ((NULL *TAGBODY-STACK*)
                (CERROR "Replace the GO with NIL" "The GO tag ~S does not appear in any enclosing TAGBODY" TAG)
                (RETURN-FROM ALPHA-GO *LITERALLY-NIL*))
               (T (CERROR "Use the tag ~*~S instead" "The GO tag ~S does not appear in any enclosing TAGBODY"
                          TAG (CAAR *TAGBODY-STACK*))
                  (SETQ DEST (CAR *TAGBODY-STACK*)))))
        (MAKE-GO :TAGBODY (CDR DEST)
                 :TAG
```

```
               (CAR DEST))))

(DEFUN ALPHA-IF (PRED-FORM THEN-FORM ELSE-FORM)
   (MAKE-IF :PRED (LET ((*CONTEXT* *PREDICATE-CONTEXT*))
                    (ALPHA-FORM PRED-FORM))
         :THEN
         (ALPHA-FORM THEN-FORM)
         :ELSE
         (ALPHA-FORM ELSE-FORM)))

(DEFUN ALPHA-IL-FUNCTION (FN CLOSE-P-FORM)

;;; If there is no close-p-form, then this is just like Common Lisp FUNCTION except that (IL:FUNCTION symbol) == 'symbol.

;;; If there is a close-p-form, then turn this into a function call, remembering to quote the close-p-form and either quote or hash-quote the function.
    ;; Account for block compilation.
   (WHEN (AND (SYMBOLP FN)
              (NOT (NULL *CURRENT-BLOCK*)))
       (LET ((LOOKUP (ASSOC FN (BLOCK-DECL-FN-NAME-MAP *CURRENT-BLOCK*))))
            (WHEN (NOT (NULL LOOKUP))                          ; This function is to be renamed.
                (SETQ FN (CDR LOOKUP)))))
   (IF (NULL CLOSE-P-FORM)
       (COND
          ((AND (SYMBOLP FN)
                (NOT (ENV-FBOUNDP *ENVIRONMENT* FN)))
           (CHECK-FOR-UNKNOWN-FUNCTION FN)
           (ALPHA-LITERAL FN))
          (T (ALPHA-FUNCTION FN)))
       (MAKE-CALL :FN (MAKE-REFERENCE-TO-VARIABLE :NAME 'IL:FUNCTION :SCOPE :GLOBAL :KIND :FUNCTION)
             :ARGS
             (LIST (IF (SYMBOLP FN)
                       (ALPHA-LITERAL FN)
                       (ALPHA-FUNCTION FN))
                   (ALPHA-LITERAL CLOSE-P-FORM)))))

(DEFUN ALPHA-LABELS (BINDINGS BODY)

;;; Make a first pass down the list of bindings in order to set up the environment in which they will all be defined.  Then alphatize each definition and
;;; transform the whole thing into a LABELS binding structure.

   (LET* ((*ENVIRONMENT* (MAKE-CHILD-ENV *ENVIRONMENT*))
          (LABELS (MAKE-LABELS))
          (STRUCTS (MAPCAR #'(LAMBDA (BINDING)
                                     (UNLESS (CHECK-ARG (CAR BINDING))
                                         (SETQ BINDING (CONS '%LOSE% (CDR BINDING))))
                                     (LET ((STRUCT (MAKE-VARIABLE :NAME (SYMBOL-NAME (CAR BINDING))
                                                                  :SCOPE :LEXICAL :KIND :FUNCTION :BINDER LABELS)))
                                          (ENV-BIND-FUNCTION *ENVIRONMENT* (CAR BINDING)
                                                 :FUNCTION STRUCT)
                                          STRUCT))
                           BINDINGS)))
        (MULTIPLE-VALUE-BIND (FORMS DECLS)
            (PARSE-BODY BODY *ENVIRONMENT* NIL)
          (BINDING-CONTOUR DECLS (UPDATE-ENVIRONMENT *ENVIRONMENT*)
                 (SETF (LABELS-FUNS LABELS)
                       (MAPCAR #'(LAMBDA (BINDING STRUCT)
                                         (CONS STRUCT (ALPHA-LAMBDA (BINDING-TO-LAMBDA BINDING)
                                                                    :NAME
                                                                    ;; Really want name to be "Foo in Bar"
                                                                    (SYMBOL-NAME (CAR BINDING)))))
                                 BINDINGS STRUCTS))
                 (SETF (LABELS-BODY LABELS)
                       (ALPHA-PROGN FORMS))))
          LABELS))

(DEFUN ALPHA-LAMBDA (ORIGINAL-FORM &KEY ((:CONTEXT *CONTEXT*)
                                         *NULL-CONTEXT*)
                                 NAME)
    ;; Check for something other than a CL:LAMBDA and coerce if necessary.
    (MULTIPLE-VALUE-BIND (FORM ARG-TYPE)
        (CONVERT-TO-CL-LAMBDA ORIGINAL-FORM)
      ;; Crack the argument list, applying any declarations that might be present.
      (LET ((ARG-LIST (SECOND FORM))
            (BODY (CDDR FORM))
            (*ENVIRONMENT* (MAKE-CHILD-ENV *ENVIRONMENT*)))
          (MULTIPLE-VALUE-BIND (CODE DECLS)
              (PARSE-BODY BODY *ENVIRONMENT* T)
            (BINDING-CONTOUR DECLS                             ; Process the declarations
                   (UPDATE-ENVIRONMENT *ENVIRONMENT*)
```

```
                    (LET* ((NODE (MAKE-LAMBDA :NAME NAME :ARG-TYPE ARG-TYPE))
                           (AUXES (ALPHA-LAMBDA-LIST ARG-LIST NODE))
                           (BODY-NODE (ALPHA-PROGN CODE)))
                          ;; AUXES is now the list of values representing the &aux variables IN REVERSE ORDER.  We must bind them around
                          ;; the body one-by-one and then wrap that in the lambda node we've already created.
                          (IL:FOR AUX IL:IN AUXES IL:DO (LET ((BINDER (MAKE-LAMBDA :REQUIRED (LIST (CAR AUX))
                                                                                   :BODY BODY-NODE)))
                                                          (SETF (VARIABLE-BINDER (CAR AUX))
                                                                BINDER)
                                                          (SETQ BODY-NODE (MAKE-CALL :FN BINDER :ARGS
                                                                                     (LIST (CDR AUX))))))
                          (SETF (LAMBDA-BODY NODE)
                                BODY-NODE)
                          ;; For Interlisp LAMBDA no-spread's, we need to save away the parameter name so that we can generate code for
                          ;; ARG properly.  (Yecch...)
                          (WHEN (EQ ARG-TYPE 2)
                              (SETF (LAMBDA-NO-SPREAD-NAME NODE)
                                    (SECOND ORIGINAL-FORM)))
                          NODE))))))
```

(DEFUN **ALPHA-LAMBDA-LIST** (ARG-LIST BINDER)

;;; Alpha-converts the argument list of a lambda form. Stores the results of the analysis into the appropriate slots of the LAMBDA structure in BINDER.
;;; Returns a list of the values representing the &aux argument variables, in reverse order of binding.

```
   (LET
    ((STATE :REQUIRED)
     REQUIRED OPTIONAL KEYWORD AUX)
    (DOLIST (ARG ARG-LIST)
        (CASE ARG
            ((&OPTIONAL) (IF (EQ STATE :REQUIRED)
                             (SETQ STATE :OPTIONAL)
                             (CERROR "Ignore it." "Misplaced &optional in lambda-list")))
            ((&REST) (IF (MEMBER STATE '(:REQUIRED :OPTIONAL))
                         (SETQ STATE :REST)
                         (CERROR "Ignore it." "Misplaced &rest in lambda-list")))
            ((&IGNORE-REST)                                          ; Internal keyword used in translation of Interlisp spread
                                                                     ; functions.
               (ASSERT (EQ STATE :OPTIONAL)
                       NIL "BUG: Misplaced &IGNORE-REST keyword.")
               (SETF (LAMBDA-REST BINDER)
                     (MAKE-VARIABLE :BINDER BINDER))
               (RETURN)                                              ; Nothing is supposed to follow an &IGNORE-REST

               )
            ((&KEY) (IF (AND (IL:NEQ STATE :AUX)
                             (IL:NEQ STATE :KEY))
                        (SETQ STATE :KEY)
                        (CERROR "Ignore it." "Misplaced &key in lambda-list")))
            ((&ALLOW-OTHER-KEYS)
               (UNLESS (EQ STATE :KEY)
                       (CERROR "Ignore it." "Stray &allow-other-keys in lambda-list."))
               (SETF (LAMBDA-ALLOW-OTHER-KEYS BINDER)
                     T))
            ((&AUX) (IF (IL:NEQ STATE :AUX)
                        (SETQ STATE :AUX)
                        (CERROR "Ignore it." "Misplaced &aux in lambda-list.")))
            (OTHERWISE (ECASE STATE
                              ((:REQUIRED) (WHEN (CHECK-ARG ARG)
                                                 (PUSH (BIND-PARAMETER ARG BINDER *ENVIRONMENT*)
                                                       REQUIRED)))
                              ((:OPTIONAL)
                                 (IF (ATOM ARG)
                                     (WHEN (CHECK-ARG ARG)
                                         (PUSH (LIST (BIND-PARAMETER ARG BINDER *ENVIRONMENT*)
                                                     *LITERALLY-NIL*)
                                               OPTIONAL))
                                     (DESTRUCTURING-BIND
                                      (VAR &OPTIONAL (INIT-FORM NIL)
                                           (SVAR NIL SV-GIVEN))
                                      ARG
                                      (WHEN (CHECK-ARG VAR)
                                          (LET ((INIT-STRUCT (ALPHA-ARGUMENT-FORM INIT-FORM)))
                                               (PUSH `(,(BIND-PARAMETER VAR BINDER *ENVIRONMENT*)
                                                       ,INIT-STRUCT
                                                       ,@(AND SV-GIVEN (CHECK-ARG SVAR)
                                                              (LIST (BIND-PARAMETER SVAR BINDER *ENVIRONMENT*))))
                                                     OPTIONAL))))))
                              ((:REST) (WHEN (CHECK-ARG ARG)
                                             (SETF (LAMBDA-REST BINDER)
                                                   (BIND-PARAMETER ARG BINDER *ENVIRONMENT*))
                                             (SETQ STATE :AFTER-REST)))
                              ((:AFTER-REST) (CERROR "Ignore it." "Stray argument ~S found after &rest var."))
                              ((:KEY)
```

```
                                (IF (ATOM ARG)
                                    (WHEN (CHECK-ARG ARG)
                                        (PUSH (LIST (INTERN (STRING ARG)
                                                            "KEYWORD")
                                                    (BIND-PARAMETER ARG BINDER *ENVIRONMENT*)
                                                    *LITERALLY-NIL*)
                                              KEYWORD))
                                    (DESTRUCTURING-BIND
                                     (KEY&VAR &OPTIONAL (INIT-FORM NIL)
                                            (SVAR NIL SV-GIVEN)
                                            &AUX KEY VAR)
                                     ARG
                                     (COND
                                        ((ATOM KEY&VAR)
                                         (WHEN (CHECK-ARG KEY&VAR)
                                             ;; This is not the real legality test; that's below.  This just makes sure that the intern will work.
                                             (SETQ KEY (INTERN (STRING KEY&VAR)
                                                               "KEYWORD")))
                                         (SETQ VAR KEY&VAR))
                                        (T (SETQ KEY (FIRST KEY&VAR))
                                           (SETQ VAR (SECOND KEY&VAR))))
                                     (WHEN (CHECK-ARG VAR)
                                         (LET ((INIT-STRUCT (ALPHA-ARGUMENT-FORM INIT-FORM)))
                                              (PUSH `(,KEY ,(BIND-PARAMETER VAR BINDER *ENVIRONMENT*)
                                                           ,INIT-STRUCT
                                                           ,@(AND SV-GIVEN (CHECK-ARG SVAR)
                                                                  (LIST (BIND-PARAMETER SVAR BINDER *ENVIRONMENT*
                                                                                        ))))
                                                    KEYWORD))))))
                                ((:AUX) (LET (VAR VAL)
                                             (COND
                                                ((ATOM ARG)
                                                 (SETQ VAR ARG)
                                                 (SETQ VAL NIL))
                                                (T (SETQ VAR (FIRST ARG))
                                                   (SETQ VAL (SECOND ARG))))
                                             (WHEN (CHECK-ARG VAR)
                                                 (LET ((TREE (ALPHA-ARGUMENT-FORM VAL)))
                                                      (PUSH (CONS (BIND-PARAMETER VAR BINDER *ENVIRONMENT*)
                                                                  TREE)
                                                            AUX)))))))))
    (SETF (LAMBDA-REQUIRED BINDER)
          (NREVERSE REQUIRED))
    (SETF (LAMBDA-OPTIONAL BINDER)
          (NREVERSE OPTIONAL))
    (SETF (LAMBDA-KEYWORD BINDER)
          (NREVERSE KEYWORD))
    AUX))
```

```
(DEFUN ALPHA-LET (BINDINGS BODY)
   ;; Install the new variables in a new environment and then install that environment before alphatizing the body.
   (MULTIPLE-VALUE-BIND (BODY DECLS)
       (PARSE-BODY BODY *ENVIRONMENT* NIL)
     (BINDING-CONTOUR DECLS (LET ((*ENVIRONMENT* (MAKE-CHILD-ENV *ENVIRONMENT*)))
                                ;; The standard is losing and wants us to install the environment before alphatizing the init-forms so that
                                ;; SPECIAL declarations will have bigger scope.  Ugh.
                                (UPDATE-ENVIRONMENT *ENVIRONMENT*)
                                (LET ((VARS NIL)
                                      (VALS NIL)
                                      (NEW-LAMBDA (MAKE-LAMBDA)))
                                     ;; Alphatize the init-forms.
                                     (IL:FOR BINDING IL:IN BINDINGS IL:DO (COND
                                                                             ((CONSP BINDING)
                                                                              (PUSH (FIRST BINDING)
                                                                                    VARS)
                                                                              (PUSH (ALPHA-ARGUMENT-FORM
                                                                                     (SECOND BINDING))
                                                                                    VALS))
                                                                             (T (PUSH BINDING VARS)
                                                                                (PUSH *LITERALLY-NIL* VALS))))
                                     ;; Bind all of the variables
                                     (SETF (LAMBDA-REQUIRED NEW-LAMBDA)
                                           (IL:FOR VAR IL:IN (NREVERSE VARS)
                                              IL:COLLECT (BIND-PARAMETER (IF (CHECK-ARG VAR)
                                                                             VAR
                                                                             '%LOSE%)
                                                                         NEW-LAMBDA *ENVIRONMENT*)))
                                     ;; Alphatize the body
                                     (SETF (LAMBDA-BODY NEW-LAMBDA)
                                           (ALPHA-PROGN BODY))
```

```
                                        (MAKE-CALL :FN NEW-LAMBDA :ARGS (NREVERSE VALS)))))))
```

(DEFUN **ALPHA-LET*** (BINDINGS BODY)

;;; Install the new variables in the environment one at a time, processing the next in an environment including those that came before. The LET* is then ;;; represented as several nested lambdas, so we must be careful to get the BINDER links set up properly.

```
   (MULTIPLE-VALUE-BIND (BODY DECLS)
       (PARSE-BODY BODY *ENVIRONMENT* NIL)
     (BINDING-CONTOUR DECLS (LET ((*ENVIRONMENT* (MAKE-CHILD-ENV *ENVIRONMENT*))
                                  (BINDING-LIST NIL))
                              (UPDATE-ENVIRONMENT *ENVIRONMENT*)
                              ;; First, alphatize each of the init-forms in the correct environment.
                              (IL:FOR BINDING IL:IN BINDINGS
                                 IL:DO (IF (CONSP BINDING)
                                           (LET ((INIT-STRUCT (ALPHA-ARGUMENT-FORM (SECOND BINDING))))
                                                (PUSH (CONS (BIND-PARAMETER (IF (CHECK-ARG (FIRST BINDING)
                                                                                           )
                                                                                (FIRST BINDING)
                                                                                '%LOSE%)
                                                                            NIL *ENVIRONMENT*)
                                                            INIT-STRUCT)
                                                      BINDING-LIST))
                                                (PUSH (CONS (BIND-PARAMETER (IF (CHECK-ARG BINDING)
                                                                                BINDING
                                                                                '%LOSE%)
                                                                            NIL *ENVIRONMENT*)
                                                            *LITERALLY-NIL*)
                                                      BINDING-LIST)))
                              ;; BINDING-LIST is now in reverse order, so we can construct the nested lambdas from the inside out.
                              (IL:BIND (BODY-STRUCT IL:_ (ALPHA-PROGN BODY)) IL:FOR PAIR IL:IN BINDING-LIST
                                 IL:DO (LET ((BINDER (MAKE-LAMBDA :REQUIRED (LIST (CAR PAIR))
                                                                  :BODY BODY-STRUCT)))
                                         (SETQ BODY-STRUCT (MAKE-CALL :FN BINDER :ARGS
                                                                      (LIST (CDR PAIR))))
                                         (SETF (VARIABLE-BINDER (CAR PAIR))
                                               BINDER))
                                 IL:FINALLY (RETURN BODY-STRUCT))))))
```

(DEFUN **ALPHA-LITERAL** (VALUE)

;;; Check for certain special values that have preallocated LITERAL structures. Otherwise, make a new one. The test for undumpable values used to be ;;; done in both COMPILE and COMPILE-FILE, but this lost in loading PCL, which COMPILE's functions containing circular structures as literals.

```
   (CASE VALUE
       ((NIL) *LITERALLY-NIL*)
       ((T) *LITERALLY-T*)
       (OTHERWISE (MAKE-LITERAL :VALUE (COND
                                          ((AND (STREAMP *INPUT-STREAM*)
                                                                ; This is COMPILE-FILE
                                                (NOT (FASL:VALUE-DUMPABLE-P VALUE)))
                                           (RESTART-CASE (ERROR "The literal value ~S would not be dumpable in a
                                                                FASL file." VALUE)
                                                  (NIL NIL :REPORT "Use the value NIL instead" NIL)
                                                  (NIL NIL :REPORT (LAMBDA (STREAM)
                                                                     (FORMAT STREAM "Use the value ~S
                                                                            anyway and hope for the best"
                                                                            VALUE))
                                                                VALUE)))
                                          (T VALUE))))))
```

(DEFUN **ALPHA-MACROLET** (BINDINGS BODY)

;;; Turn the bindings into expansion functions and add them into the environment for the analysis of the body.

```
   (LET ((NEW-ENV (MAKE-CHILD-ENV *ENVIRONMENT*)))
        (IL:FOR MACRO IL:IN BINDINGS IL:DO (ENV-BIND-FUNCTION NEW-ENV (CAR MACRO)
                                                              :MACRO
                                                              (CRACK-DEFMACRO (CONS 'DEFMACRO MACRO))))
        (LET ((*ENVIRONMENT* NEW-ENV))
             (MULTIPLE-VALUE-BIND (FORMS DECLS)
                 (PARSE-BODY BODY *ENVIRONMENT* NIL)
               (BINDING-CONTOUR DECLS (UPDATE-ENVIRONMENT *ENVIRONMENT*)
                      (ALPHA-PROGN FORMS))))))
```

```
(DEFUN ALPHA-MV-CALL (FN-FORM ARG-FORMS)
   (LET (VALUES-USED)
        (MULTIPLE-VALUE-BIND (FN NOT-INLINE?)
            (ALPHA-FUNCTIONAL-FORM FN-FORM)
          (COND
             ((AND (NULL (CDR ARG-FORMS))
```

```
                  (LAMBDA-P FN)
                  (NOT (OR (LAMBDA-OPTIONAL FN)
                           (LAMBDA-REST FN)
                           (LAMBDA-KEYWORD FN))))                ; In this very common case, we can tell how many values are
                                                                 ; expected.
             (SETQ VALUES-USED (LENGTH (LAMBDA-REQUIRED FN))))
            (T (SETQ VALUES-USED :UNKNOWN)))
          (IF (NULL ARG-FORMS)                                   ; This is silly, but we'd better handle it correctly.
              (MAKE-CALL :FN FN :ARGS NIL :NOT-INLINE NOT-INLINE?)
              (MAKE-MV-CALL :FN FN :ARG-EXPRS (LET ((*CONTEXT* (MAKE-CONTEXT :VALUES-USED VALUES-USED)))
                                                (MAPCAR #'ALPHA-FORM ARG-FORMS))
                    :NOT-INLINE NOT-INLINE?)))))

(DEFUN ALPHA-MV-PROG1 (FORMS)
   (LET ((VALS-USED (CONTEXT-VALUES-USED *CONTEXT*)))
        (COND
           ((NULL (CDR FORMS))
            (ALPHA-FORM (CAR FORMS)))
           ((AND (NUMBERP VALS-USED)
                 (< VALS-USED 2))                                ; The multiple values aren't wanted.  Make this a normal PROG1.
            (ALPHA-FORM (CONS 'PROG1 FORMS)))
           (T (MAKE-MV-PROG1 :STMTS (CONS (ALPHA-FORM (FIRST FORMS))
                                          (LET ((*CONTEXT* *EFFECT-CONTEXT*))
                                               (MAPCAR #'ALPHA-FORM (REST FORMS)))))))))

(DEFUN ALPHA-PROGN (FORMS)
   (IF (NULL (CDR FORMS))
       (ALPHA-FORM (CAR FORMS))
       (MAKE-PROGN :STMTS (LET ((OLD-CONTEXT *CONTEXT*)
                                (*CONTEXT* *EFFECT-CONTEXT*))
                            (IL:FOR TAIL IL:ON FORMS IL:COLLECT (IF (NULL (CDR TAIL))
                                                                    (LET ((*CONTEXT* OLD-CONTEXT))
                                                                         (ALPHA-FORM (CAR TAIL)))
                                                                    (ALPHA-FORM (CAR TAIL))))))))

(DEFUN ALPHA-PROGV (SYMS-EXPR VALS-EXPR BODY-FORMS)
   (MAKE-PROGV :SYMS-EXPR (ALPHA-ARGUMENT-FORM SYMS-EXPR)
          :VALS-EXPR
          (ALPHA-ARGUMENT-FORM VALS-EXPR)
          :STMT
          (ALPHA-PROGN BODY-FORMS)))

(DEFUN ALPHA-RETURN-FROM (NAME FORM)
   (LET ((DEST (ASSOC NAME *BLOCK-STACK*)))
        (WHEN (NULL DEST)
            (COND
               ((NULL *BLOCK-STACK*)
                (CERROR "Treat (RETURN-FROM name value-form) as simply value-form" "~S, found in a RETURN-FROM,
                       is not the name of any enclosing BLOCK" NAME)
                (RETURN-FROM ALPHA-RETURN-FROM (ALPHA-FORM FORM)))
               (T (CERROR "Use the name ~*~S instead" "~S, found in a RETURN-FROM, is not the name of any
                         enclosing BLOCK" NAME (CAAR *BLOCK-STACK*))
                  (SETQ DEST (CAR *BLOCK-STACK*)))))
        (MAKE-RETURN :BLOCK (CDR DEST)
               :VALUE
               (LET ((*CONTEXT* (BLOCK-CONTEXT (CDR DEST))))
                    (ALPHA-FORM FORM))))))

(DEFUN ALPHA-SETQ (KIND FORMS)
   (LET ((SETQS (IL:FOR TAIL IL:ON FORMS IL:BY (CDDR TAIL) IL:COLLECT (WHEN (AND (EQ KIND 'SETQ)
                                                                                 (NULL (CDR TAIL)))
                                                                        (CERROR "Add an extra NIL on the end
                                                                                 of the form" "Odd number of
                                                                                 forms given to SETQ."))
                                                                      (MAKE-SETQ :VAR (RESOLVE-VARIABLE-REFERENCE
                                                                                       *ENVIRONMENT*
                                                                                       (CAR TAIL)
                                                                                       T)
                                                                                 :VALUE
                                                                                 (ALPHA-ARGUMENT-FORM (CADR TAIL)))))))
        (IF (NULL (CDR SETQS))
            (CAR SETQS)
            (MAKE-PROGN :STMTS SETQS))))

(DEFUN ALPHA-TAGBODY (BODY)

;;; Break up the body into 'segments', each of which is an unbroken series of forms along with the zero or more tags that begin that series of forms.

   (WHEN (NULL BODY)
         (RETURN-FROM ALPHA-TAGBODY *LITERALLY-NIL*))
   (LET ((TAGBODY (MAKE-TAGBODY))
```

```
          (*TAGBODY-STACK* *TAGBODY-STACK*))
        ;; Make a first pass down the body to find all of the tags
        (IL:FOR FORM IL:IN BODY IL:DO (WHEN (ATOM FORM)
                                            (PUSH (CONS FORM TAGBODY)
                                                  *TAGBODY-STACK*)))
        ;; On the second pass, put together the segments and alphatize all of the forms
        (DO ((*CONTEXT* *EFFECT-CONTEXT*)
             (SEGMENT-LIST NIL))
            ((NULL BODY)
             (SETF (TAGBODY-SEGMENTS TAGBODY)
                   (NREVERSE SEGMENT-LIST)))
          (LET ((SEGMENT (MAKE-SEGMENT)))
               (DO NIL
                   ((OR (NULL BODY)
                        (CONSP (CAR BODY))))
                 (PUSH (POP BODY)
                       (SEGMENT-TAGS SEGMENT)))
               (DO ((FORM-LIST NIL))
                   ((OR (NULL BODY)
                        (ATOM (CAR BODY)))
                    (SETF (SEGMENT-STMTS SEGMENT)
                          (NREVERSE FORM-LIST)))
                 (PUSH (ALPHA-FORM (POP BODY))
                       FORM-LIST))
               (PUSH SEGMENT SEGMENT-LIST)))
        TAGBODY))

(DEFUN ALPHA-THROW (TAG VALUE)
  (MAKE-THROW :TAG (ALPHA-ARGUMENT-FORM TAG)
              :VALUE
              (LET ((*CONTEXT* *NULL-CONTEXT*))
                   (ALPHA-FORM VALUE))))

(DEFUN ALPHA-UNWIND-PROTECT (BODY CLEANUPS)
  (MAKE-UNWIND-PROTECT :STMT (ALPHA-LAMBDA (LET ((CLEANUP-VAR (GENSYM)))
                                                `(LAMBDA (,CLEANUP-VAR)
                                                         (MULTIPLE-VALUE-PROG1 ,BODY (FUNCALL ,CLEANUP-VAR))))
                                            :CONTEXT *CONTEXT* :NAME 'SI::*UNWIND-PROTECT*)
              :CLEANUP
              (ALPHA-LAMBDA `(LAMBDA NIL ,@CLEANUPS)
                            :CONTEXT *EFFECT-CONTEXT* :NAME "Clean-up forms")))

(DEFUN CONVERT-TO-CL-LAMBDA (FORM)
  ;; Return two values: a CL:LAMBDA form equivalent to the given one and the Interlisp ARGTYPE for the form.
  (CASE (CAR FORM)
        ((LAMBDA)
         ;; Common Lisp LAMBDA's have indeterminate ARGTYPE.  The assembler will figure out whether it's 0 or 2.  The LOCALVARS
         ;; declaration is because Interlisp's scoping rules have overwhelmed those of Common Lisp, may they rest in peace.
         (VALUES `(LAMBDA ,(SECOND FORM)
                          (DECLARE (IL:LOCALVARS . T))
                          ,@(CDDR FORM))
                 NIL))
        ((IL:LAMBDA IL:OPENLAMBDA) (IF (LISTP (SECOND FORM))
                                       ;; LAMBDA spread.  Use the Common Lisp &OPTIONAL keyword and also one made for internal
                                       ;; compiler use that will throw away the extra arguments.
                                       (VALUES `(LAMBDA (&OPTIONAL ,@(SECOND FORM)
                                                                   &IGNORE-REST)
                                                        ,@(CDDR FORM))
                                               0)
                                       ;; LAMBDA no-spread.  Bind the parameter to the number of arguments passed.  The handling of
                                       ;; ARG must be done in code generation, unfortunately.
                                       (VALUES `(LAMBDA NIL (LET ((,(SECOND FORM)
                                                                   (IL:\\MYARGCOUNT)))
                                                              ,@(CDDR FORM)))
                                               2)))
        ((IL:NLAMBDA) (IF (LISTP (SECOND FORM))
                          ;; NLAMBDA spread.  Just like the LAMBDA-spread case but we have a different ARG-TYPE.
                          (VALUES `(LAMBDA (&OPTIONAL ,@(SECOND FORM)
                                                      &IGNORE-REST)
                                           ,@(CDDR FORM))
                                  1)
                          ;; NLAMBDA no-spread.  We take exactly one argument and are otherwise entirely normal.
                          (VALUES `(LAMBDA (,(SECOND FORM))
                                           ,@(CDDR FORM))
                                  3)))
        (OTHERWISE
```

```
          ;; This is not my beautiful LAMBDA form!
          (CERROR "Use (LAMBDA () NIL) instead" "The form ~S should be a LAMBDA form but is not." FORM)
          (VALUES '(LAMBDA NIL NIL)
                 0))))

(DEFUN COMPLETELY-EXPAND (FORM)
   (IF (ATOM FORM)
       FORM
       (LET ((NEW-FORM FORM)
             CHANGED-P)
            (IL:UNTIL (MEMBER (CAR NEW-FORM)
                              '(BLOCK CATCH
                                   COMPILER-LET
                                   DECLARE
                                   EVAL-WHEN
                                   FLET
                                   IL:FUNCTION
                                   FUNCTION
                                   GO
                                   IF
                                   LABELS
                                   LET
                                   LET*
                                   MACROLET
                                   SI::%MACROLET
                                   MULTIPLE-VALUE-CALL
                                   MULTIPLE-VALUE-PROG1
                                   PROGN
                                   PROGV
                                   QUOTE
                                   SETQ
                                   IL:SETQ
                                   TAGBODY
                                   THE
                                   THROW
                                   UNWIND-PROTECT)
                              :TEST
                              'EQ)
              IL:DO (MULTIPLE-VALUE-SETQ (NEW-FORM CHANGED-P)
                        (OPTIMIZE-AND-MACROEXPAND-1 NEW-FORM))
                    (WHEN (NULL CHANGED-P)
                        (IF (AND (CONSP (CAR NEW-FORM))
                                 (EQ 'IL:OPENLAMBDA (CAAR NEW-FORM)))
                            (SETQ NEW-FORM (EXPAND-OPENLAMBDA-CALL (CAR NEW-FORM)
                                                (CDR NEW-FORM)))
                            (RETURN NEW-FORM)))
              IL:FINALLY (RETURN NEW-FORM)))))

(DEFUN EXPAND-OPENLAMBDA-CALL (FN ARGS)

;;; The idea here is to try to do some substitution into the body of the OPENLAMBDA.  We do it here instead of in meta-evaluation because there are
;;; parts of the Interlisp system that count on their optimizers being able to find literals in their arguments.  They count on the substitution being done so
;;; that that will be the case.

;;; It is well-known that the use of SUBLIS here is a bug: for example, if one of the arguments to the OPENLAMBDA has the same name as one of the
;;; functions called therein, the subst will still change both of them, undoubtably leading to chaos.  However, the ByteCompiler has always done it this
;;; way and nothing broke, so, since it's also very easy, we do it too.  If anything actually counts on this, though, I may kill the author.

;;; The general details of this transformation are the way they are because it's the way the ByteCompiler did it.  Pavel will never defend this code on
;;; philosophical grounds.  ("If this code is caught or killed, Pavel will disavow any knowledge of its actions...")

   (LET ((UNSUBBED-PARAMS NIL)
         (UNSUBBED-ARGS NIL)
         (SUBST-ALIST NIL)
         EXTRA-ARGS)
        (DO* ((PARAMS (CADR FN)
                      (CDR PARAMS))
              (ARGS (LET ((*CONTEXT* *ARGUMENT-CONTEXT*))
                         (MAPCAR 'COMPLETELY-EXPAND ARGS))
                    (CDR ARGS))
              (ARG (CAR ARGS)
                   (CAR ARGS)))
             ((NULL PARAMS)
              (SETQ EXTRA-ARGS ARGS))
          ;; For each pair, if the argument is a constant, add it to the substitution we'll later apply.
          (COND
             ((OR (CONSTANTP ARG)
                  (AND (ATOM ARG)
                       (NOT (SYMBOLP ARG)))
                  (AND (CONSP ARG)
                       (EQ (CAR ARG)
                           'IL:FUNCTION)
                       (SYMBOLP (CADR ARG))))
```

```
            (PUSH (CONS (CAR PARAMS)
                        ARG)
                  SUBST-ALIST))
           (T (PUSH (CAR PARAMS)
                    UNSUBBED-PARAMS)
              (PUSH ARG UNSUBBED-ARGS))))
     (WHEN (NULL UNSUBBED-ARGS)                                    ; We got rid of all of them.
         (RETURN-FROM EXPAND-OPENLAMBDA-CALL `(PROGN ,@EXTRA-ARGS ,@(SUBLIS SUBST-ALIST (CDDR FN)
                                                                          :TEST
                                                                          'EQ))))
```

;; Perhaps there're no extra arguments or they're all constants. This should really be a full-blown test for side-effect freedom, but that's too ;; much work for alphatization.

```
     (COND
        ((AND EXTRA-ARGS (NOTEVERY #'(LAMBDA (ARG)
                                       (OR (CONSTANTP ARG)
                                           (AND (ATOM ARG)
                                                (NOT (SYMBOLP ARG)))
                                           (AND (CONSP ARG)
                                                (MEMBER (CAR ARG)
                                                        '(IL:FUNCTION FUNCTION)))))))
```

;; There're extra arguments in the way, so we're done.

```
         (SETF (CAR UNSUBBED-ARGS)
               `(PROG1 ,(CAR UNSUBBED-ARGS)
                       ,@EXTRA-ARGS))
         `((LAMBDA ,(REVERSE UNSUBBED-PARAMS)
                   ,@(SUBLIS SUBST-ALIST (CDDR FN)
                             :TEST
                             'EQ))
           ,@(REVERSE UNSUBBED-ARGS)))
        (T
```

;; There's nothing interesting between the body and the as yet unsubbed arguments, so maybe we can also substitute some ;; variables. Note that because the unsubbed lists are in reverse order now, we can easily examine the arguments starting with the ;; last one and working backwards, just as we'd like.

```
           (IL:WHILE (AND UNSUBBED-ARGS (SYMBOLP (FIRST UNSUBBED-ARGS)))
              IL:DO (PUSH (CONS (POP UNSUBBED-PARAMS)
                                (POP UNSUBBED-ARGS))
                          SUBST-ALIST))
           (COND
              ((NULL UNSUBBED-ARGS)                                ; All substituted in.
               `(PROGN ,@(SUBLIS SUBST-ALIST (CDDR FN)
                                 :TEST
                                 'EQ)))
              ((MEMBER (CAR (FIRST UNSUBBED-ARGS))
                       '(IL:SETQ SETQ))
               (COND
                  ((NULL (CDR UNSUBBED-ARGS))
                   (PUSH (CONS (FIRST UNSUBBED-PARAMS)
                               (CADR (FIRST UNSUBBED-ARGS)))
                         SUBST-ALIST)
                   `(PROGN ,(FIRST UNSUBBED-ARGS)
                           ,@(SUBLIS SUBST-ALIST (CDDR FN)
                                     :TEST
                                     'EQ)))
                  (T (PUSH (CONS (POP UNSUBBED-PARAMS)
                                 (CADR (FIRST UNSUBBED-ARGS)))
                           SUBST-ALIST)
                     (SETQ UNSUBBED-ARGS (CONS `(PROG1 ,(SECOND UNSUBBED-ARGS)
                                                       ,(FIRST UNSUBBED-ARGS))
                                               (CDDR UNSUBBED-ARGS)))
                     `((LAMBDA ,(REVERSE UNSUBBED-PARAMS)
                               ,@(SUBLIS SUBST-ALIST (CDDR FN)
                                         :TEST
                                         'EQ))
                       ,@(REVERSE UNSUBBED-ARGS)))))
              (T `((LAMBDA ,(REVERSE UNSUBBED-PARAMS)
                           ,@(SUBLIS SUBST-ALIST (CDDR FN)
                                     :TEST
                                     'EQ))
                   ,@(REVERSE UNSUBBED-ARGS)))))))
```

;; Alphatization testing

```
(DEFPARAMETER *INDENT-INCREMENT* 3
```

;;; Number of spaces by which the indentation should increase in nested nodes.

```
   )


(DEFVAR *NODE-HASH* NIL
   "Used by the parse-tree pretty-printer")

(DEFVAR *NODE-NUMBER* 0
```

```
   "Used by the parse-tree pretty-printer")


(DEFUN TEST-ALPHA (FN)
   (LET ((TREE (TEST-ALPHA-2 FN)))
        (UNWIND-PROTECT
            (PRINT-TREE TREE)
            (RELEASE-TREE TREE))))


(DEFUN TEST-ALPHA-2 (FN)
   (LET ((*ENVIRONMENT* (MAKE-ENV))
         (*CONTEXT* *NULL-CONTEXT*)
         (*CONSTANTS-HASH-TABLE* (MAKE-HASH-TABLE))
         (IL:SPECVARS T)
         (IL:LOCALVARS IL:SYSLOCALVARS)
         (IL:GLOBALVARS IL:GLOBALVARS)
         (IL:LOCALFREEVARS NIL)
         (*PROCESSED-FUNCTIONS* NIL)
         (*UNKNOWN-FUNCTIONS* NIL)
         (*CURRENT-FUNCTION* NIL)
         (*AUTOMATIC-SPECIAL-DECLARATIONS* NIL))
        (DECLARE (SPECIAL IL:SPECVARS IL:LOCALVARS IL:LOCALFREEVARS IL:GLOBALVARS))
        (ALPHA-LAMBDA (COND
                          ((CONSP FN)
                           FN)
                          ((CONSP (IL:GETD FN))
                           (IL:GETD FN))
                          (T (PARSE-DEFUN (IL:GETDEF FN 'IL:FUNCTIONS)))))))


(DEFUN PARSE-DEFUN (FORM)
   (DESTRUCTURING-BIND (IGNORE NAME ARG-LIST &BODY BODY)
          FORM
          (MULTIPLE-VALUE-BIND (FORMS DECLS)
              (PARSE-BODY BODY NIL T)
            `(LAMBDA ,ARG-LIST ,@DECLS (BLOCK ,NAME ,@FORMS)))))


(DEFUN PRINT-TREE (TREE)
   (LET ((*NODE-HASH* (MAKE-HASH-TABLE))
         (*NODE-NUMBER* 0)
         (*PRINT-CASE* :UPCASE))
        (PRINT-NODE TREE 0))
   (TERPRI)
   (VALUES))


(DEFUN PRINT-NODE (NODE INDENT)

;;; NODE is the node to print.  INDENT is the number of spaces over we are on entry to PRINT-NODE.  We should not ever print anything on the line to
;;; the left of that point.

   (LET ((NUMBER (AND (NOT (LITERAL-P NODE))
                      (GETHASH NODE *NODE-HASH*))))
        (COND
           (NUMBER (FORMAT T "-~S-" NUMBER))
           (T (INCF *NODE-NUMBER*)
              (SETF (GETHASH NODE *NODE-HASH*)
                    *NODE-NUMBER*)
              (FORMAT T "~S. ~A: " *NODE-NUMBER* (TYPE-OF NODE))
              (LET ((NESTED-INDENT (+ INDENT *INDENT-INCREMENT*)))
                   (MACROLET ((NEW-LINE (&OPTIONAL (DELTA 0))
                                        `(FORMAT T "~%~vT" (+ NESTED-INDENT ,DELTA)))
                              (PRINT-BLIPPER-INFO NIL '(FORMAT T "  Closed-over-p: ~:[false~;true~]
                                                                 New-frame-p: ~:[false~;true~]" (
                                                                                                 BLIPPER-CLOSED-OVER-P
                                                                                                   NODE)
                                                                 (BLIPPER-NEW-FRAME-P NODE))))
                            (ETYPECASE NODE
                                (BLOCK-NODE
                                   (PRIN1 (BLOCK-NAME NODE))
                                   (PRINT-BLIPPER-INFO)
                                   (NEW-LINE)
                                   (PRINT-NODE (BLOCK-STMT NODE)
                                          NESTED-INDENT))
                                (CALL-NODE
                                   (WHEN (CALLER-NOT-INLINE NODE)
                                         (PRINC "(not inline)"))
                                   (NEW-LINE)
                                   (PRINC "Func: ")
                                   (PRINT-NODE (CALL-FN NODE)
                                          (+ NESTED-INDENT 6))
                                   (WHEN (CALL-ARGS NODE)
                                       (NEW-LINE)
                                       (PRINC "Args: ")
                                       (IL:FOR ARG-TAIL IL:ON (CALL-ARGS NODE)
```

```
                IL:DO (PRINT-NODE (CAR ARG-TAIL)
                          (+ NESTED-INDENT 6))
                      (WHEN (NOT (NULL (CDR ARG-TAIL)))
                            (NEW-LINE 6)))))
(CATCH-NODE
   (NEW-LINE)
   (PRINC "Tag:  ")
   (PRINT-NODE (CATCH-TAG NODE)
          (+ NESTED-INDENT 6))
   (NEW-LINE)
   (PRINC "Stmt: ")
   (PRINT-NODE (CATCH-STMT NODE)
          (+ NESTED-INDENT 6)))
(GO-NODE
   (FORMAT T "to ~S" (GO-TAG NODE))
   (NEW-LINE)
   (PRINC "Tagbody: ")
   (PRINT-NODE (GO-TAGBODY NODE)
          (+ NESTED-INDENT 9)))
(IF-NODE
   (NEW-LINE)
   (PRINC "Pred: ")
   (PRINT-NODE (IF-PRED NODE)
          (+ NESTED-INDENT 6))
   (NEW-LINE)
   (PRINC "Then: ")
   (PRINT-NODE (IF-THEN NODE)
          (+ NESTED-INDENT 6))
   (NEW-LINE)
   (PRINC "Else: ")
   (PRINT-NODE (IF-ELSE NODE)
          (+ NESTED-INDENT 6)))
(LABELS-NODE
   (NEW-LINE)
   (PRINC "Funs: ")
   (IL:FOR TAIL IL:ON (LABELS-FUNS NODE)
      IL:DO (PRINT-NODE (CAAR TAIL)
                 (+ NESTED-INDENT 6))
            (NEW-LINE 10)
            (PRINT-NODE (CDAR TAIL)
                   (+ NESTED-INDENT 10))
            (WHEN (NOT (NULL (CDR TAIL)))
                  (NEW-LINE 6)))
   (NEW-LINE)
   (PRINC "Body: ")
   (PRINT-NODE (LABELS-BODY NODE)
          (+ NESTED-INDENT 6)))
(LAMBDA-NODE
   (NEW-LINE)
   (WHEN (LAMBDA-REQUIRED NODE)
       (PRINC "&req: ")
       (IL:FOR VARS IL:ON (LAMBDA-REQUIRED NODE)
          IL:DO (PRINT-NODE (CAR VARS)
                         (+ NESTED-INDENT 6))
                (IF (NULL (CDR VARS))
                    (NEW-LINE)
                    (NEW-LINE 6))))
   (WHEN (LAMBDA-OPTIONAL NODE)
       (PRINC "&opt: ")
       (IL:FOR VARS IL:ON (LAMBDA-OPTIONAL NODE)
          IL:DO (DESTRUCTURING-BIND (VAR &OPTIONAL (INIT NIL I-GIVEN)
                                                   (SVAR NIL SV-GIVEN))
                                    (CAR VARS)
                                    (COND
                                       ((SYMBOLP VAR)
                                        (PRINT-NODE (CAR VARS)
                                                    (+ NESTED-INDENT 6)))
                                       ((NOT I-GIVEN)
                                        (PRINT-NODE VAR (+ NESTED-INDENT 6)))
                                       (T (PRINC "(")
                                          (PRINT-NODE VAR (+ NESTED-INDENT 7))
                                          (NEW-LINE 7)
                                          (PRINT-NODE INIT (+ NESTED-INDENT 7))
                                          (NEW-LINE 7)
                                          (WHEN SV-GIVEN
                                              (PRINT-NODE SVAR (+ NESTED-INDENT 7))
                                              (NEW-LINE 7))
                                          (PRINC ")"))))
                (IF (NULL (CDR VARS))
                    (NEW-LINE)
                    (NEW-LINE 6))))
   (WHEN (LAMBDA-REST NODE)
       (PRINC "&rest: ")
       (PRINT-NODE (LAMBDA-REST NODE)
              (+ NESTED-INDENT 7))
       (NEW-LINE))
   (WHEN (LAMBDA-KEYWORD NODE)
```

```
                (PRINC "&key: ")
                (IL:FOR VARS IL:ON (LAMBDA-KEYWORD NODE)
                   IL:DO (DESTRUCTURING-BIND (KEY VAR &OPTIONAL (INIT NIL I-GIVEN)
                                                    (SVAR NIL SV-GIVEN))
                                (CAR VARS)
                                (FORMAT T "((~S " KEY)
                                (NEW-LINE 8)
                                (PRINT-NODE VAR (+ NESTED-INDENT 8))
                                (PRINC ")")
                                (NEW-LINE 7)
                                (PRINT-NODE INIT (+ NESTED-INDENT 7))
                                (NEW-LINE 7)
                                (WHEN SV-GIVEN
                                    (PRINT-NODE SVAR (+ NESTED-INDENT 7))
                                    (NEW-LINE 7))
                                (PRINC ")"))
                         (COND
                            ((NULL (CDR VARS))
                             (WHEN (LAMBDA-ALLOW-OTHER-KEYS NODE)
                                   (PRINC "&allow-other-keys"))
                             (NEW-LINE))
                            (T (NEW-LINE 6)))))
            (WHEN (LAMBDA-CLOSED-OVER-VARS NODE)
                (PRINC "Closed-over:")
                (NEW-LINE 10)
                (IL:FOR VARS IL:ON (LAMBDA-CLOSED-OVER-VARS NODE)
                   IL:DO (PRINT-NODE (CAR VARS)
                                 (+ NESTED-INDENT 10))
                         (IF (NULL (CDR VARS))
                             (NEW-LINE)
                             (NEW-LINE 10))))
            (PRINT-NODE (LAMBDA-BODY NODE)
                  NESTED-INDENT))
        (LITERAL-NODE (PRIN1 (LITERAL-VALUE NODE)))
        (MV-CALL-NODE
            (WHEN (CALLER-NOT-INLINE NODE)
                  (PRINC "(not inline)"))
            (NEW-LINE)
            (PRINC "Func: ")
            (PRINT-NODE (MV-CALL-FN NODE)
                   (+ NESTED-INDENT 6))
            (NEW-LINE)
            (PRINC "Args: ")
            (IL:FOR ARG-TAIL IL:ON (MV-CALL-ARG-EXPRS NODE)
               IL:DO (PRINT-NODE (CAR ARG-TAIL)
                             (+ NESTED-INDENT 6))
                     (WHEN (NOT (NULL (CDR ARG-TAIL)))
                           (NEW-LINE 6))))
        (MV-PROG1-NODE (IL:FOR STMT IL:IN (MV-PROG1-STMTS NODE)
                          IL:DO (NEW-LINE)
                                (PRINT-NODE STMT NESTED-INDENT)))
        (OPCODES-NODE (PRIN1 (OPCODES-BYTES NODE)))
        (PROGN-NODE (IL:FOR STMT IL:IN (PROGN-STMTS NODE) IL:DO (NEW-LINE)
                                                                (PRINT-NODE STMT
                                                                       NESTED-INDENT)))
        (PROGV-NODE
            (NEW-LINE)
            (PRINC "Vars: ")
            (PRINT-NODE (PROGV-SYMS-EXPR NODE)
                   (+ NESTED-INDENT 6))
            (NEW-LINE)
            (PRINC "Vals: ")
            (PRINT-NODE (PROGV-VALS-EXPR NODE)
                   (+ NESTED-INDENT 6))
            (NEW-LINE)
            (PRINC "Body: ")
            (PRINT-NODE (PROGV-STMT NODE)
                   (+ NESTED-INDENT 6)))
        (RETURN-NODE
            (NEW-LINE)
            (PRINC "From:  ")
            (PRINT-NODE (RETURN-BLOCK NODE)
                   (+ NESTED-INDENT 7))
            (NEW-LINE)
            (PRINC "Value: ")
            (PRINT-NODE (RETURN-VALUE NODE)
                   (+ NESTED-INDENT 7)))
        (SETQ-NODE
            (NEW-LINE)
            (PRINC "Var:   ")
            (PRINT-NODE (SETQ-VAR NODE)
                   (+ NESTED-INDENT 7))
            (NEW-LINE)
            (PRINC "Value: ")
            (PRINT-NODE (SETQ-VALUE NODE)
                   (+ NESTED-INDENT 7)))
        (TAGBODY-NODE
```

```
                                (PRINT-BLIPPER-INFO)
                                (IL:FOR SEGMENT IL:IN (TAGBODY-SEGMENTS NODE)
                                   IL:DO (IL:FOR TAG IL:IN (SEGMENT-TAGS SEGMENT) IL:DO (NEW-LINE)
                                                                                        (PRINC TAG))
                                         (IL:FOR STMT IL:IN (SEGMENT-STMTS SEGMENT)
                                            IL:DO (NEW-LINE 4)
                                                  (PRINT-NODE STMT (+ NESTED-INDENT 4)))))
                             (THROW-NODE
                                (NEW-LINE)
                                (PRINC "Tag:   ")
                                (PRINT-NODE (THROW-TAG NODE)
                                       (+ NESTED-INDENT 7))
                                (NEW-LINE)
                                (PRINC "Value: ")
                                (PRINT-NODE (THROW-VALUE NODE)
                                       (+ NESTED-INDENT 7)))
                             (UNWIND-PROTECT-NODE
                                (NEW-LINE)
                                (PRINC "Stmt:    ")
                                (PRINT-NODE (UNWIND-PROTECT-STMT NODE)
                                       (+ NESTED-INDENT 9))
                                (NEW-LINE)
                                (PRINC "Cleanup: ")
                                (PRINT-NODE (UNWIND-PROTECT-CLEANUP NODE)
                                       (+ NESTED-INDENT 9)))
                             ((OR VARIABLE-STRUCT VAR-REF-NODE) (LET ((VAR (IF (VARIABLE-P NODE)
                                                                               NODE
                                                                               (VAR-REF-VARIABLE NODE))))
                                                                  (FORMAT T "~S ~S ~S ~@[~*Closed-over ~]"
                                                                         (VARIABLE-SCOPE VAR)
                                                                         (VARIABLE-KIND VAR)
                                                                         (VARIABLE-NAME VAR)
                                                                         (VARIABLE-CLOSED-OVER VAR))
                                                                  (WHEN (VARIABLE-BINDER VAR)
                                                                      (COND
                                                                         ((GETHASH (VARIABLE-BINDER VAR)
                                                                                 *NODE-HASH*)
                                                                          (PRINC "Binder: ")
                                                                          (PRINT-NODE (VARIABLE-BINDER VAR)
                                                                                 0))
                                                                         (T (NEW-LINE)
                                                                            (PRINC "Binder: ")
                                                                            (PRINT-NODE (VARIABLE-BINDER
                                                                                          VAR)
                                                                                   (+ NESTED-INDENT 8)))))))
)))))))

(DEFPARAMETER CONTEXT-TEST-FORM
   '(PROGN (CTXT)
           (LIST (IF (CTXT)
                     (CTXT))
                 (MULTIPLE-VALUE-LIST (CTXT))
                 (MULTIPLE-VALUE-CALL #'(LAMBDA (A B)
                                                (BAR A B))
                        (CTXT))
                 (MULTIPLE-VALUE-CALL #'(LAMBDA (A &REST B)
                                                (BAR A B))
                        (CTXT))
                 (MULTIPLE-VALUE-CALL #'(LAMBDA (A B)
                                                (BAR A B))
                        (CTXT)
                        (CTXT))
                 (LET ((X (CTXT)))
                      (SETQ X (CTXT)))
                 ((LAMBDA (A &OPTIONAL (B (CTXT)))
                          (CTXT))
                  (CTXT))
                 (MULTIPLE-VALUE-CALL #'(LAMBDA (A B)
                                                (BAR A B))
                        ((LAMBDA (C)
                                 (CTXT))
                         17)))
           (CTXT))
   "Form for testing the alphatizer's manipulation of context information.")

(DEFMACRO CTXT ()
   (PRINC-TO-STRING *CONTEXT*))

;; Arrange to use the correct compiler.

(IL:PUTPROPS IL:XCLC-ALPHA IL:FILETYPE COMPILE-FILE)

;; Arrange for the correct makefile environment
```

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

FUNCTION INDEX

ALPHA-ARGUMENT-FORM 3
ALPHA-ATOM 3
ALPHA-BLOCK 3
ALPHA-CATCH 4
ALPHA-COMBINATION 4
ALPHA-COMPILER-LET 4
ALPHA-EVAL-WHEN 4
ALPHA-FLET 4
ALPHA-FORM 5
ALPHA-FUNCTION 6
ALPHA-FUNCTIONAL-FORM ...6
ALPHA-GO 6
ALPHA-IF 7
ALPHA-IL-FUNCTION 7
ALPHA-LABELS 7
ALPHA-LAMBDA 7
ALPHA-LAMBDA-LIST 8
ALPHA-LET 9
ALPHA-LET* 10
ALPHA-LITERAL 10
ALPHA-MACROLET 10
ALPHA-MV-CALL 10
ALPHA-MV-PROG1 11
ALPHA-PROGN 11
ALPHA-PROGV 11
ALPHA-RETURN-FROM 11
ALPHA-SETQ 11
ALPHA-TAGBODY 11
ALPHA-THROW 12
ALPHA-UNWIND-PROTECT ...12
BIND-PARAMETER 3
BINDING-TO-LAMBDA 3
CHECK-ARG 3
COMPLETELY-EXPAND 13
CONVERT-TO-CL-LAMBDA ...12
EXPAND-OPENLAMBDA-CALL .13
PARSE-DEFUN 15
PRINT-NODE 15
PRINT-TREE 15
PROCESS-DECLARATIONS 1
PROCESS-IL-DECLARATIONS .2
TEST-ALPHA 15
TEST-ALPHA-2 15
UPDATE-ENVIRONMENT 3

VARIABLE INDEX

BLOCK-STACK 3
INDENT-INCREMENT 14
NODE-HASH 14
NODE-NUMBER 14
TAGBODY-STACK 3
CONTEXT-TEST-FORM 18

MACRO INDEX

BINDING-CONTOUR 1
CTXT 18

PROPERTY INDEX

IL:XCLC-ALPHA 18,19