This document should be referenced as:

Dearle, A. “A Persistent Architecture Intermediate Language”. Universities of Glasgow and St Andrews Technical Report PPRR-35-87 (1987).

A Persistent Architecture Intermediate Language

Alan Dearle

11th August 1986

PAIL structures

The intermediate code may be partitioned into thirteen categories. These are:

1) Basic tree structure
2) Symbol table entries
3) Control
4) Assignment
5) Store Allocation
6) Indexing
7) Aliasing
8) Scoping
9) Store to Store operations
10) Literals
11) Application
12) Comments
13) Optimisations

These partitions will now be examined fully in turn.

1. Basic tree structure.

PAIL code trees are typed. Pointers to parent nodes in the code tree are also provided in order that traversers can access entire code trees from an arbitrary position. Generated code will contain pointers into the trees. **Note** that in this document *code* will mean some arbitary PAIL code and *tree* will mean a an object of the class described below. Type,code and parent node information is bound together in a structure of the following class:

structure tree(pntr Type,Code,Parent)
where

Type is a pointer to an encoding of the type of the subtree pointed at by Code.
Code is a pointer to the PAIL code.
Parent is a pointer to the parent tree node.

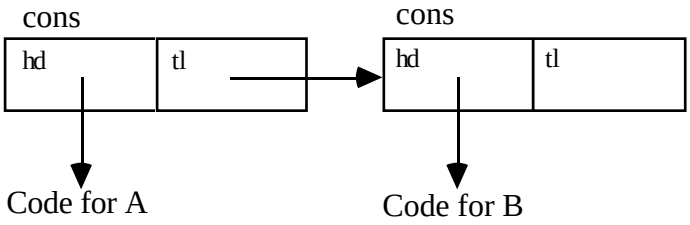
The following class is used to build lists of things:

structure cons(pntr hd,tl)
where

hd is a pointer to some PAIL code.
tl is a pointer to another cons structure or nil.

PS-algol source: A,B

PAIL code:



2. Symbol table Entries

```
structure link ( cstring name ;
                 pntr type.info, initial.value ;
                 bool manifest, retained, primitive ;
                 pntr location.info )
```

where

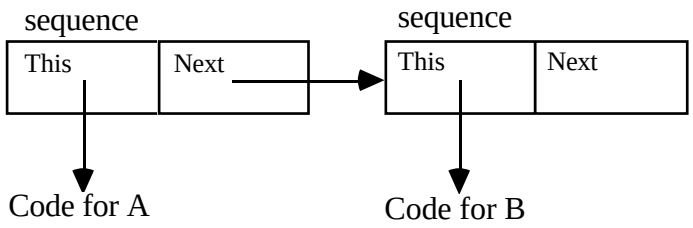
name is the name of the identifier.
type.info holds an encoding of the type.
initial.value will contain a syntax tree for the initialising expression.
manifest has the value true if value is known at compile time.(Not set in V2.1)
retained has the value true if the object is retained in a block.(Not set in V2.1)
primitive has the value true if the object is a special function.
location.info is filled in by the code generator.

3. Control

structure sequence(pntr This,Next)
where
This is a pointer to some code.
Next is a pointer to another sequence or nil

PS-algol source: A ; B

PAIL code:

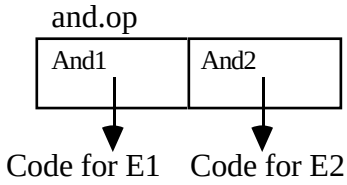


structure and.op(pntr And1,And2)
where
And1 is a pointer to the code for the first operand of and.
And2 is a pointer to the code for the second operand of and.

Note that *and* and *or* are in this section because they are non-strict in their arguments.

PS-algol source: E1 and E2

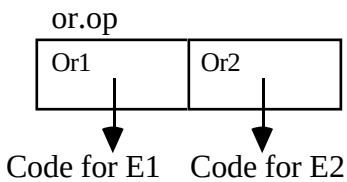
PAIL code:



structure or.op(pntr Or1,Or2)
where
Or1 is a pointer to the code for the first operand of or.
Or2 is a pointer to the code for the second operand of or.

PS-algol source: E1 or E2

PAIL code:

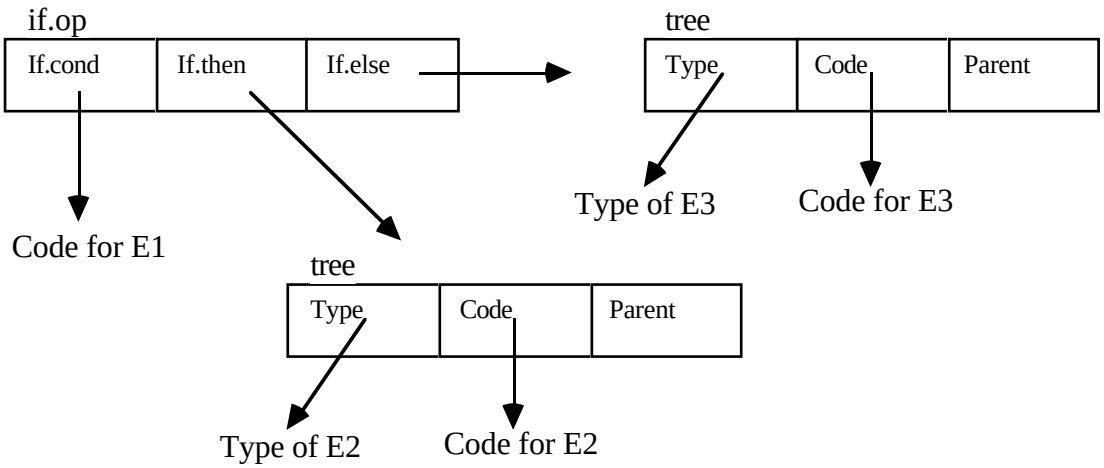


structure if.op(pntr If.cond,If.then,If.else)
where

If.cond is a pointer to the code for the boolean condition.
If.then is a pointer to the tree for the then branch.
If.else is a pointer to the tree for the else branch.

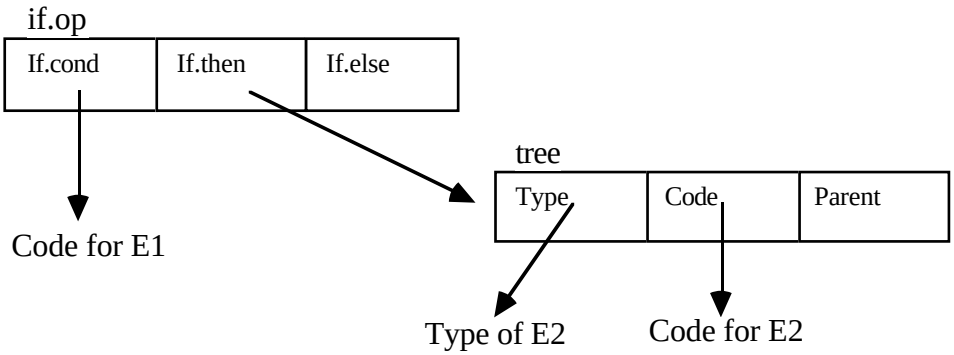
PS-algol source: if E1 then E2 else E3

PAIL code:



PS-algol source: if E1 do E2

PAIL code:

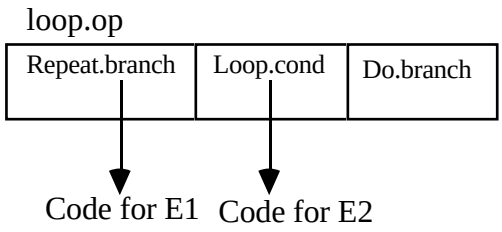


structure loop.op(pntr Repeat.branch,Loop.cond,Do.branch)
where

Repeat.branch is a pointer to the code for the unconditionally repeated part.
Loop.cond is a pointer to the code for the boolean condition.
Do.branch is a pointer to the code for the conditionally repeated part.

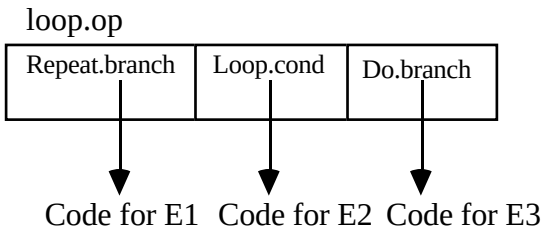
PS-algol source: repeat E1 while E2

PAIL code:



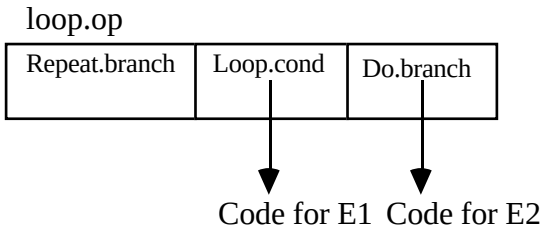
PS-algol source: repeat E1 while E2 do E3

PAIL code:



PS-algol source: while E1 do E2

PAIL code:



structure for.op(pntr For.symbol.table,
For.iterator For.set,For.do)

where

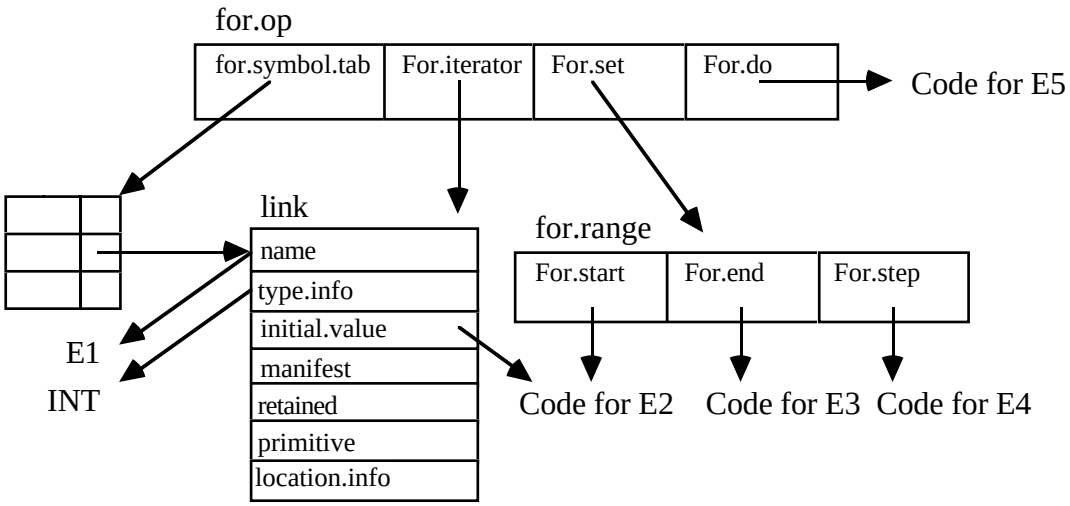
For.symbol.table is a pointer to the block symbol table containing the iterator.
For.iterator is a pointer to the control variable symbol table entry.
For.do is a pointer to the repeated code.
For.set is a pointer to an instance of the following structure class:

structure for.range(pntr For.start,For.end,For.step)
where

For.start is a pointer to the initialising code.
For.end is a pointer to the code for the loop end value.
For.step is a pointer to the code for the loop increment.

PS-algol source: for E1 = E2 to E3 by E4 do E5

PAIL code:



structure case.op(pntr Case.switch,Choice.list,Default)
where

Case.switch is a pointer to the tree for the switch expression.
Default is a pointer to the tree for the default expression.
Choice.list is a cons list of the following structures:

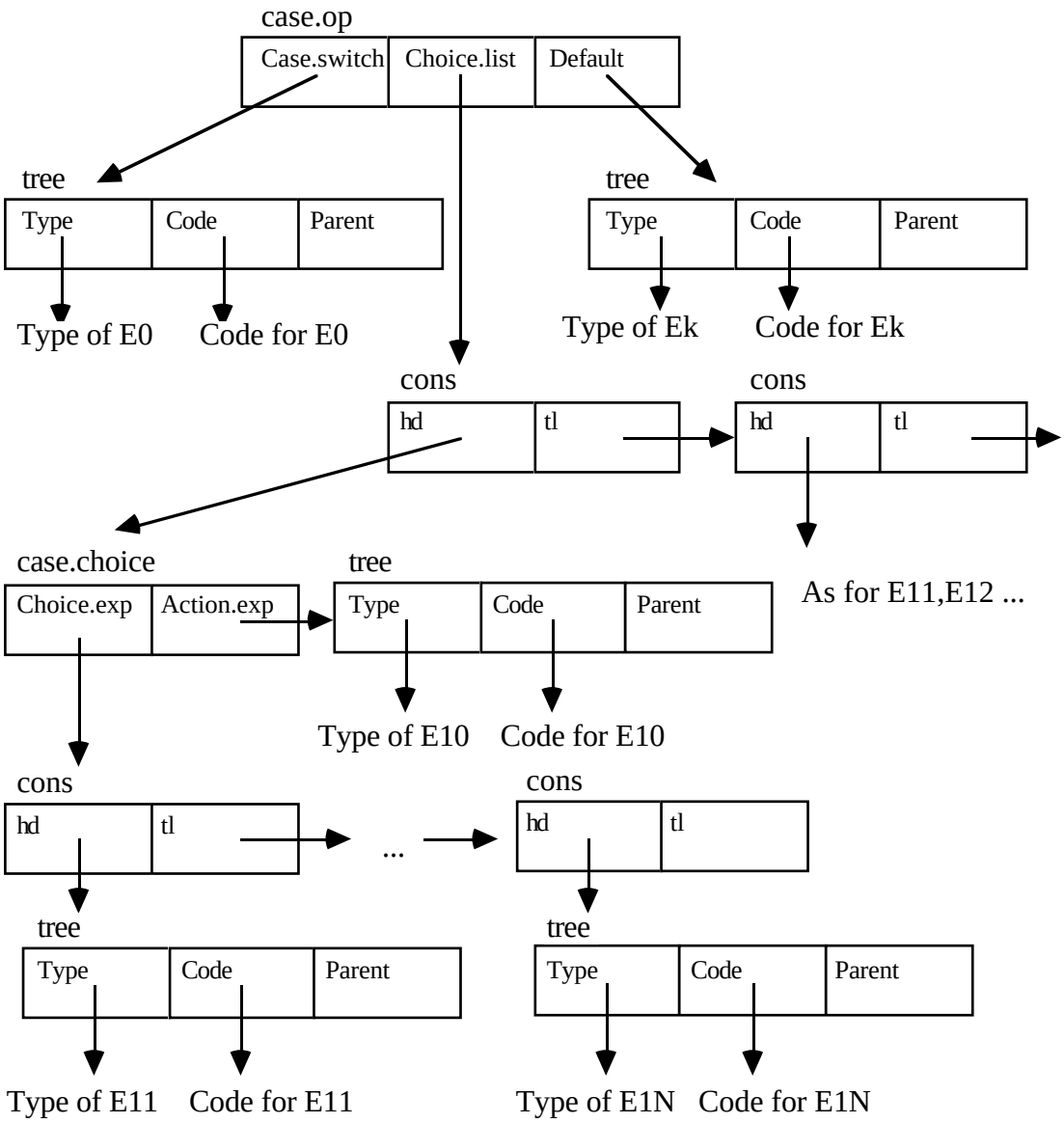
structure case.choice(pntr Choice.exp,Action.exp)
where

Choice.exp is a cons list of pointers to the trees for the selectors.
Action.exp is a pointer to the tree for the case action

PS-algol source:

```
case E0 of
E11,E12,...E1n      : E10
......
Ej1,Ej2,...Ejn   : Ej0
default          : Ek0
```

PAIL code:



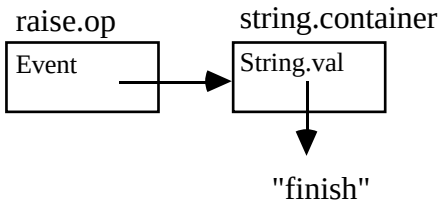
structure raise.op(pntr Event)
where

Event is a pointer to the code for an event.

Note that initially the only two events that we have are abort and end of program. These will be represented by two strings,"finish" and "abort".

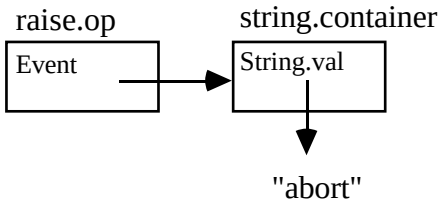
PS-algol source: ?

PAIL code:



PS-algol source: abort

PAIL code:



structure catch.op(pntr Handler,Protected.code)
where

Handler is a pointer to the code for an event handler.
Protected.code is a pointer to the code which the handler handles.

No PS-algol code currently generates this PAIL structure.

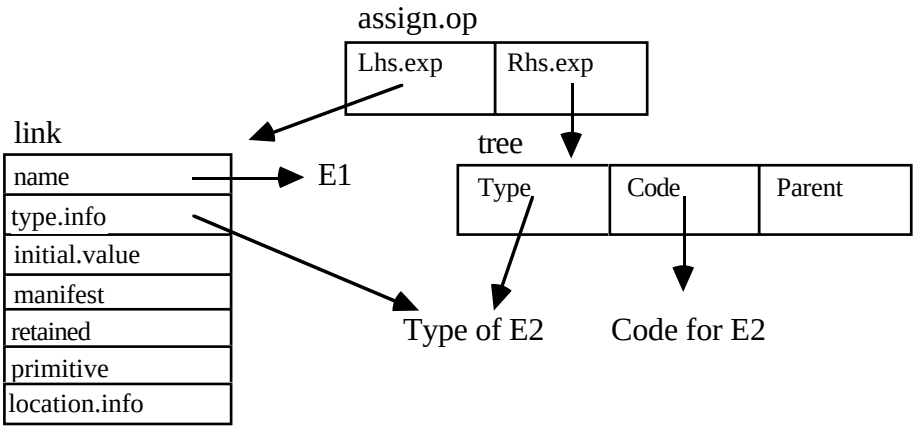
4. Assignment

structure assign.op(pntr Lhs.exp,Rhs.exp)
where

Lhs.exp is a pointer to the code representing an address.
Note that this may be a link or a tree (in the case of sub.addr.op).
Rhs is the tree for a value.

PS-algol source: E1 := E2

PAIL code:



5. Store Allocation

structure iliffe.op(pntr Vec.init,Bounds.list)
where

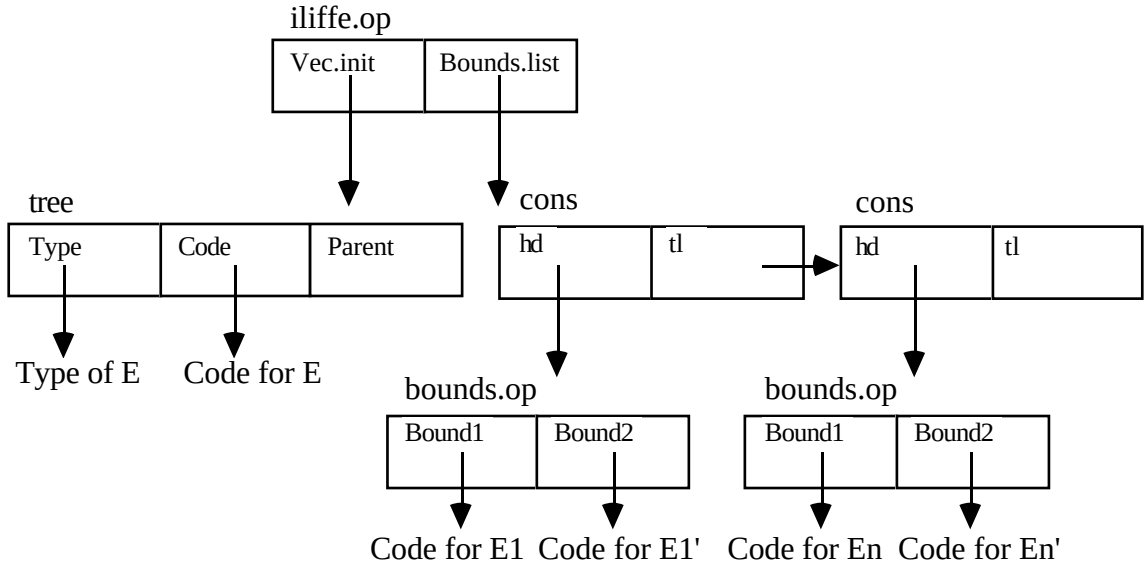
Vec.init is a pointer to the tree for the initialising expression.
Bounds.list is a cons list of the following structures:

structure bounds.op(pntr Bound1,Bound2)
where

Bound1 contains a pointer to the code for the lower bound of the sub vector.
Bound2 contains a pointer to the code for the upper bound of the sub vector.

PS-algol source: vector E1::E1',...En::En' of E

PAIL code:

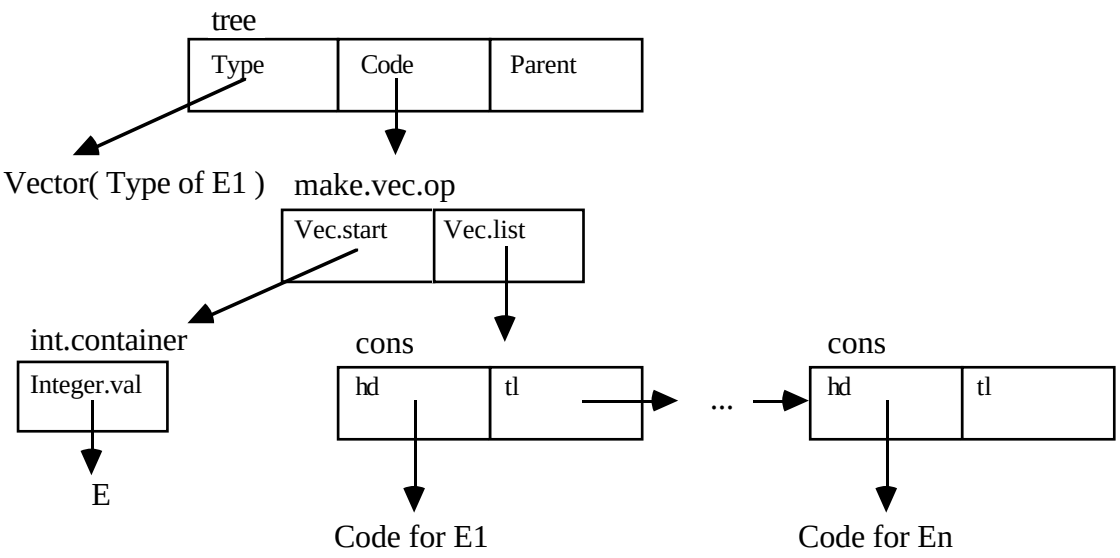


structure make.vec.op(pntr Vec.start,Vec.list)
where

Vec.start is a pointer to the code for the lower bound of the vector.
Vec.list is a cons list of code for the vector elements.

PS-algol source: @E of T[E1,....En]

PAIL code:

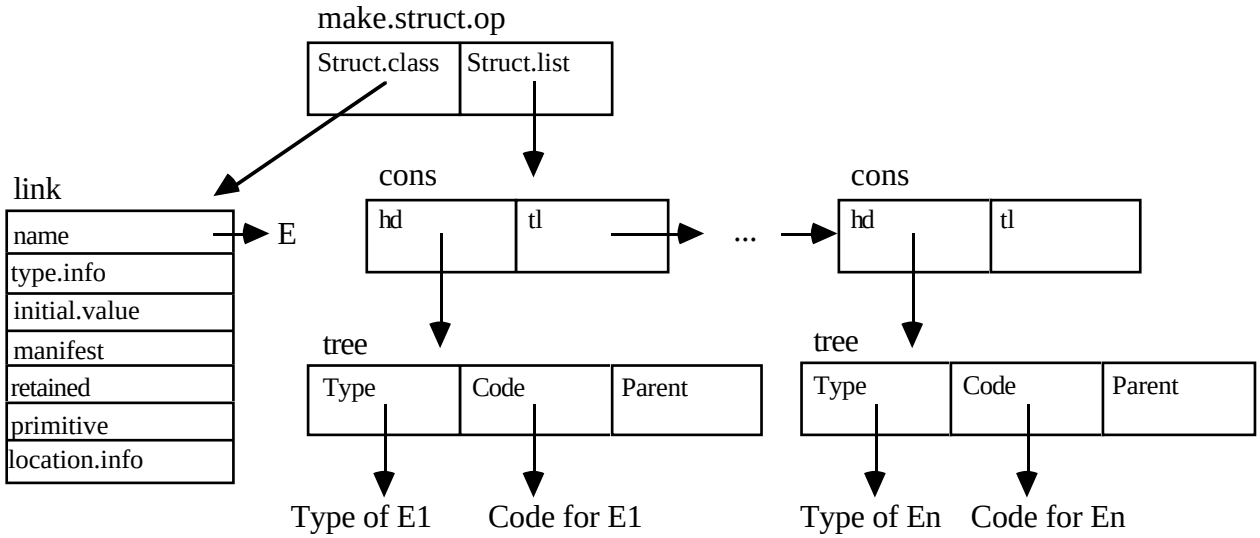


structure make.struct.op(pntr Struct.class,Struct.list)
where

Struct.class is a pointer to the symbol table entry for the structure.
Struct.list is a cons list of the trees for the elements.

PS-algol source: E(E1,.....En)

PAIL code:

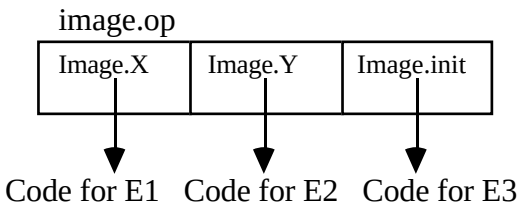


structure image.op(pntr Image.X,Image.Y,Image.init)
where

Image.X is a pointer to the code for the code for the images X dimension
Image.Y is a pointer to the code for the code for the images Y dimension
Image.init is a pointer to the code for the code for the images initial `colour'

PS-algol source: image E1 by E2 of E3

PAIL code:

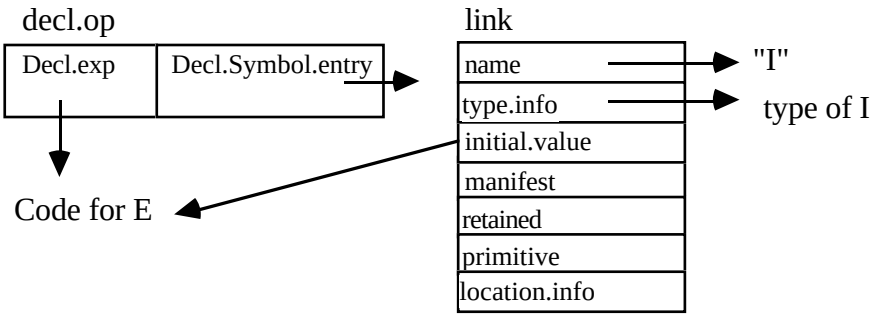


structure decl.op(pntr Decl.exp,Decl.Symbol.entry)
where

Decl.exp is a pointer to the code for the initialising expression.
Decl.Symbol.entry is the symbol table entry for the identifier.

PS-algol source: let I = E
let I := E

PAIL code:



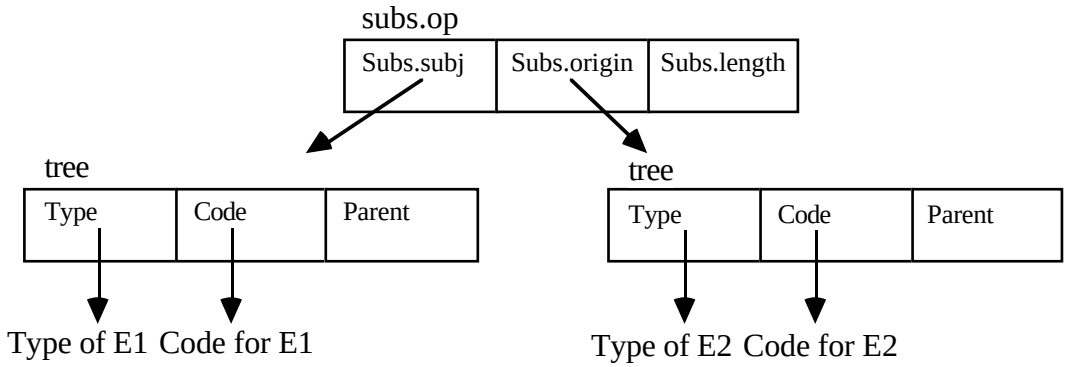
6. Indexing

structure subs.op(pntr Subs.subject,Subs.origin,Subs.length)
where

Subs.subject is a pointer to the tree for the object.
Subs.origin is a pointer to the tree for the offset into the object.
Subs.length is a pointer to the tree for the range of the index.

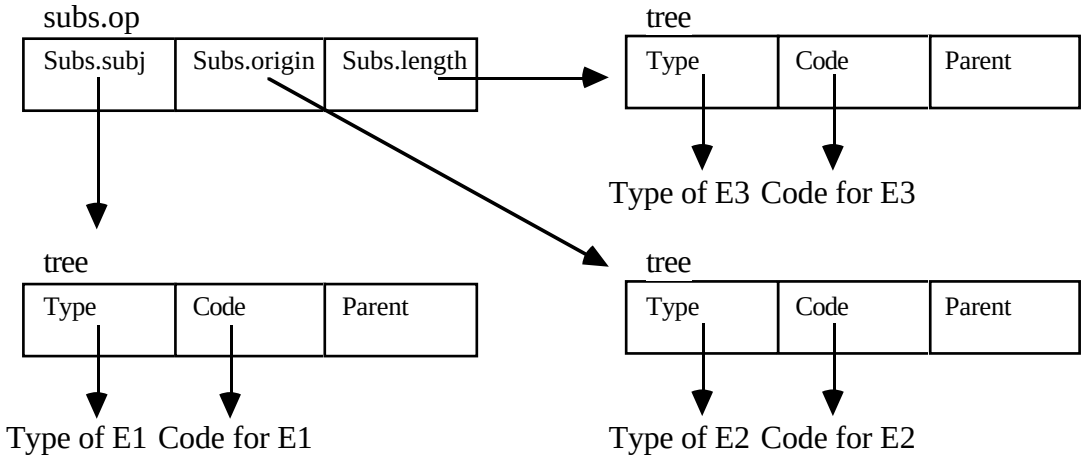
PS-algol source: E1(E2)

PAIL code:



PS-algol source: E1(E2|E3)

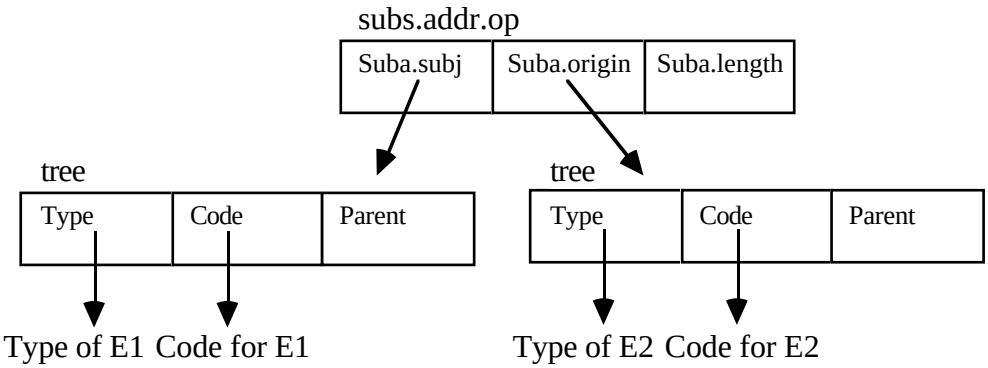
PAIL code:



structure subs.addr.op(pntr Suba.subject,Suba.origin,Suba.length)
where
Suba.subject is a pointer to the tree for the object.
Suba.origin is a pointer to the tree for the offset into the object.
Suba.length is a pointer to the tree for the range of the index.
whenever
The index yields a location rather than a value.

PS-algol source: E1(E2)

PAIL code:



7. Aliasing

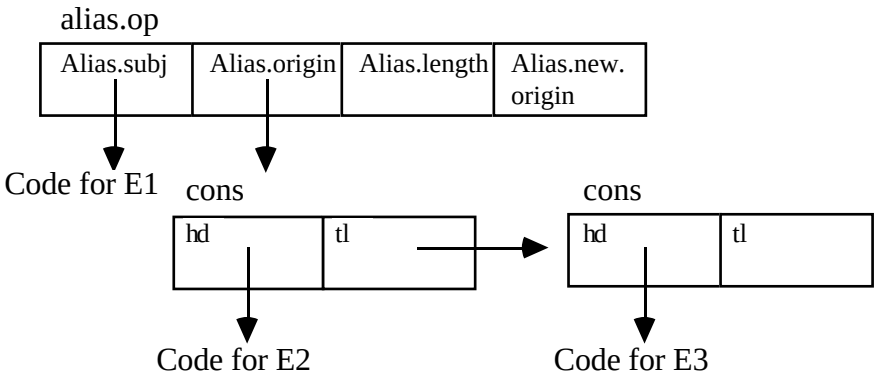
structure alias.op(pntr Alias.subject,Alias.origin,
Alias.length,Alias.new.origin)

where

Alias.subject is a pointer to the code for the object.
Alias.origin is a cons list of the code for the offset into the object.
Alias.length is a cons list of the code for the range of the index.
Alias.new.origin is a pointer to code for the origin of the alias.

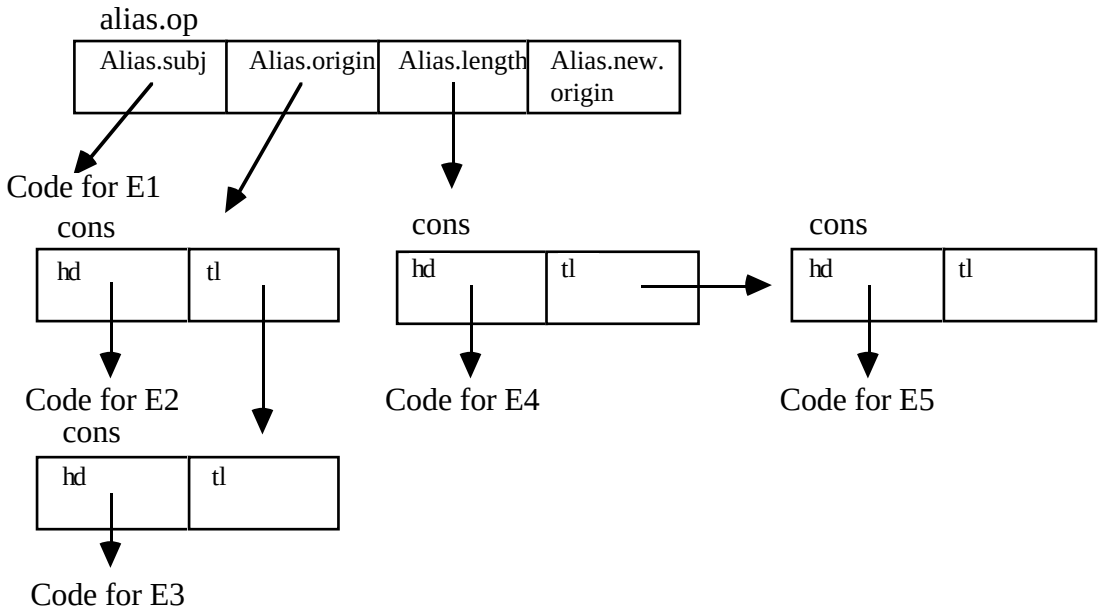
PS-algol source: limit E1 at E2,E3

PAIL code:



PS-algol source: limit E1 to E2 by E3 at E4,E5

PAIL code:



8. Scoping

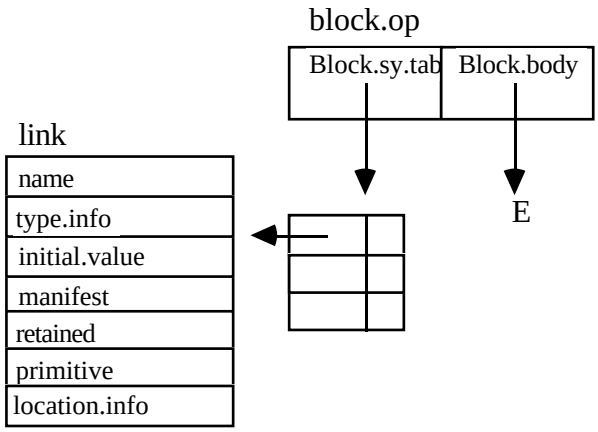
structure block.op(pntr Block.symbol.table,Block.body)
where

Block.symbol.table is a pointer to the symbol table for the block.
Block.body is a pointer to the code for the block.

PS-algol source: { E }

PS-algol source: begin E end

PAIL code:



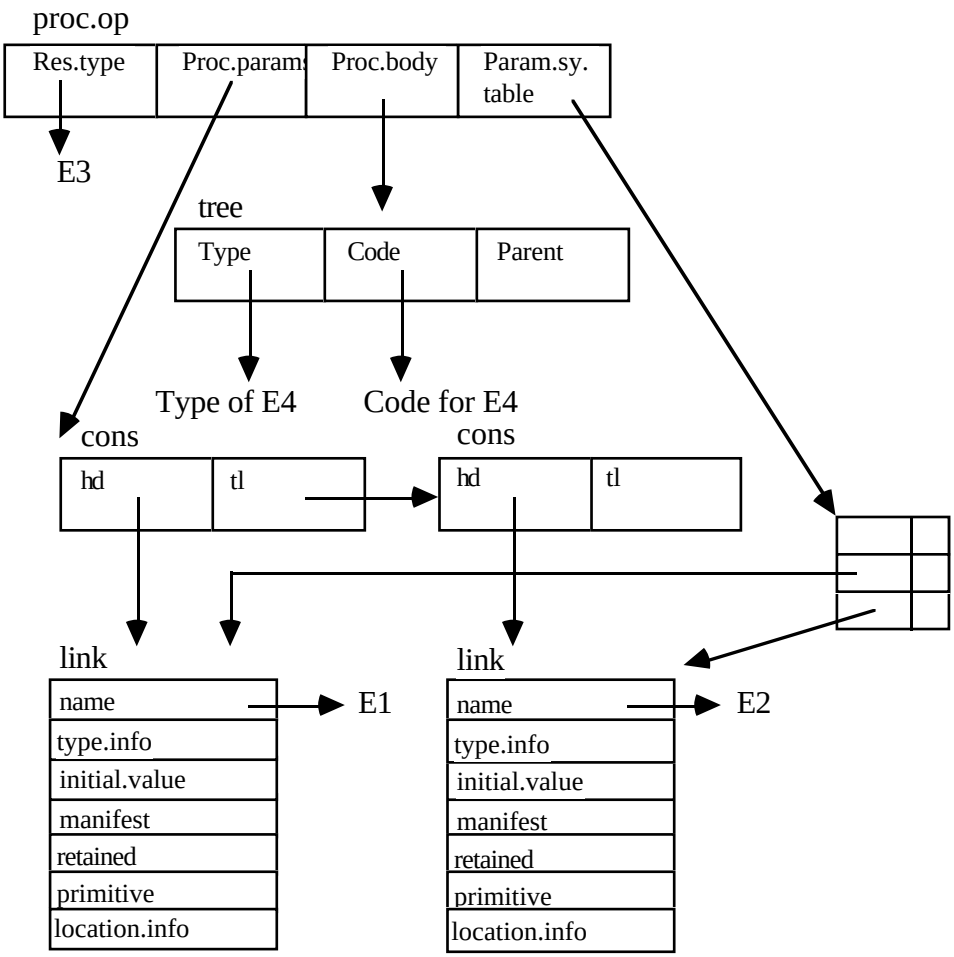
structure proc.op(pntr Res.type,Proc.params,
Proc.body,Param.symb.table)

where

Res.type is the result type.
Proc.params is a cons list of symbol table entries for the procedure parameters.
Proc.body is a pointer to the tree for the procedure body.
Param.symb.table is the symbol table for the parameters

PS-algol source: proc(E1,E2 -> E3) ; E4

PAIL code:



9. Store to Store operations

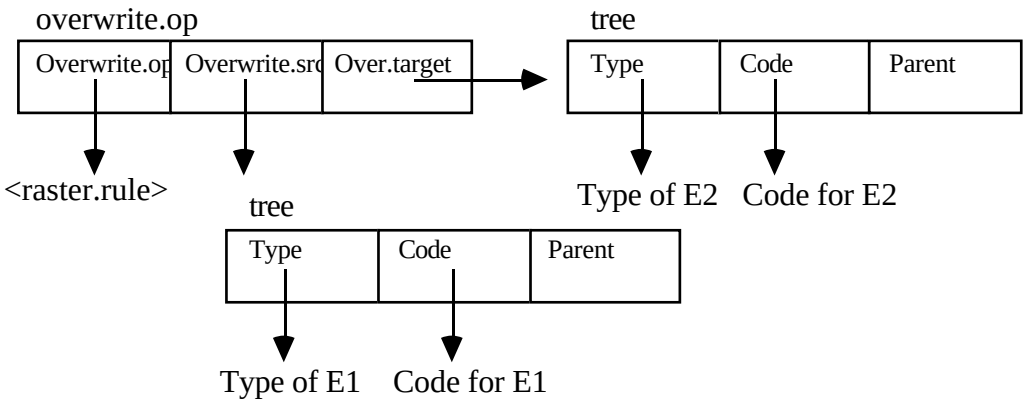
structure overwrite.op(string Overwrite.op ;
pntr Overwrite.src,Overwrite.target)

where

Overwrite.op indicates which raster op it is
Overwrite.src is a pointer to the tree for the source image
Overwrite.target is a pointer to the tree for the target image

PS-algol source: <raster-rule> E1 onto E2

PAIL code:

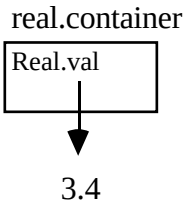


10. Literals

structure real.container(real Real.val)

PS-algol source: 3.4

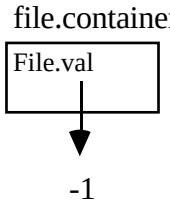
PAIL code:



structure file.container(int File.val)

PS-algol source: nullfile

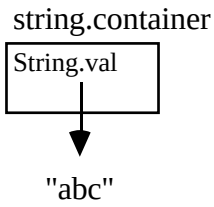
PAIL code:



structure string.container(string String.val)

PS-algol source: "abc"

PAIL code:

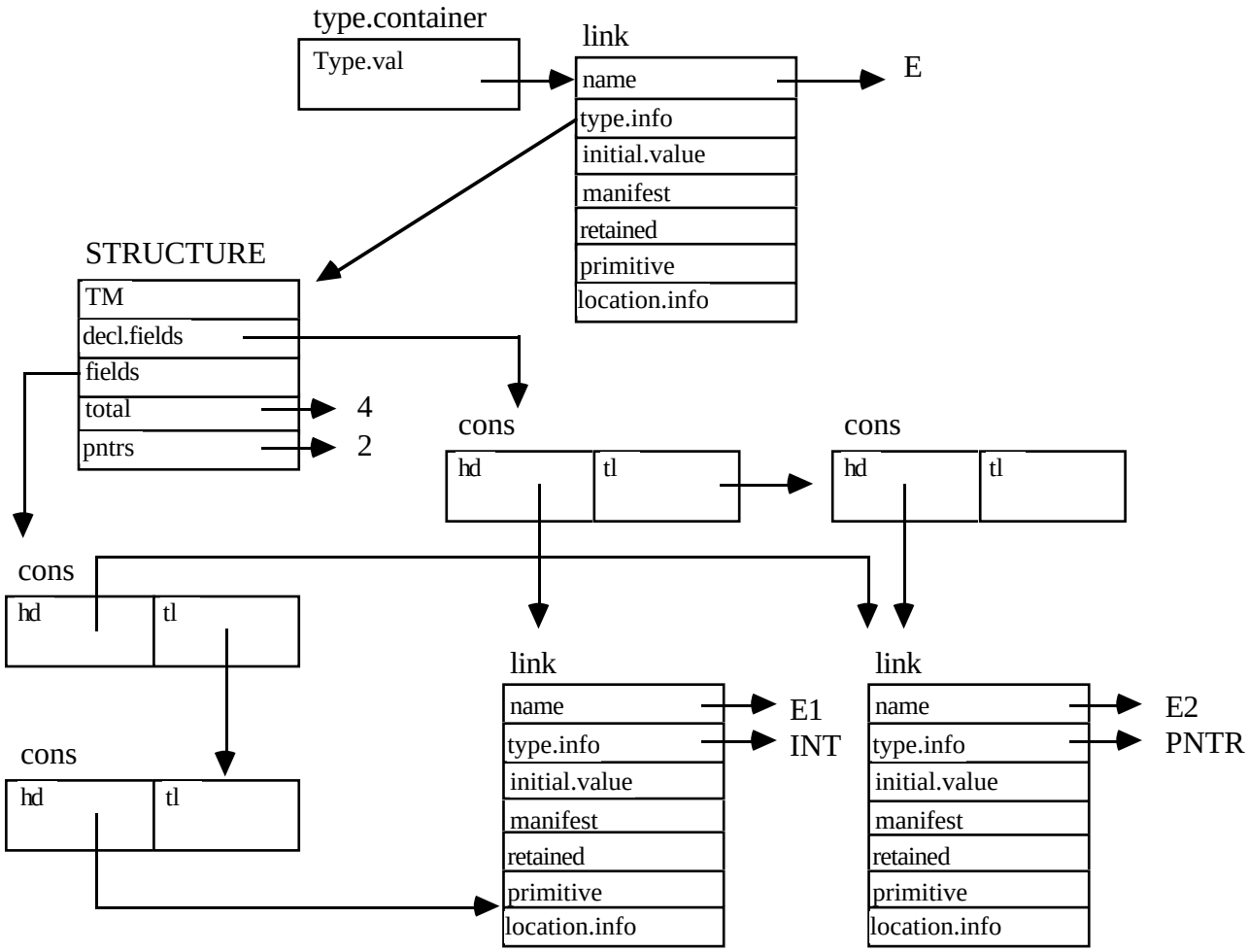


structure type.container(pntr Type.val)

PS-algol source: structure E(int E1 ; pntr E2)

Note that in the type description for STRUCTURE *decl.fields* holds information on the structure fields in declaration order whereas *fields* holds the same information in normalised order with the pointer values first. Please see the abstract machine manual for more details.

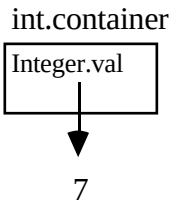
PAIL code:



structure int.container(int Integer.val)

PS-algol source: 7

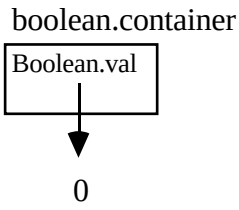
PAIL code:



structure boolean.container(int Boolean.val)

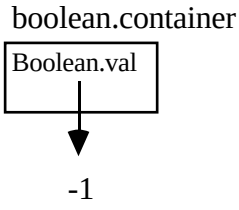
PS-algol source: true

PAIL code:



PS-algol source: false

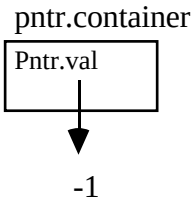
PAIL code:



structure pntr.container(int Pntr.val)

PS-algol source: nil

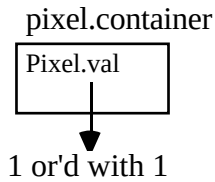
PAIL code:



structure pixel.container(int Pixel.val)

PS-algol source: on

PAIL code:



Note that the pixel representation is the same as that used in the abstract machine - see abstract machine manual for details.

11. Application

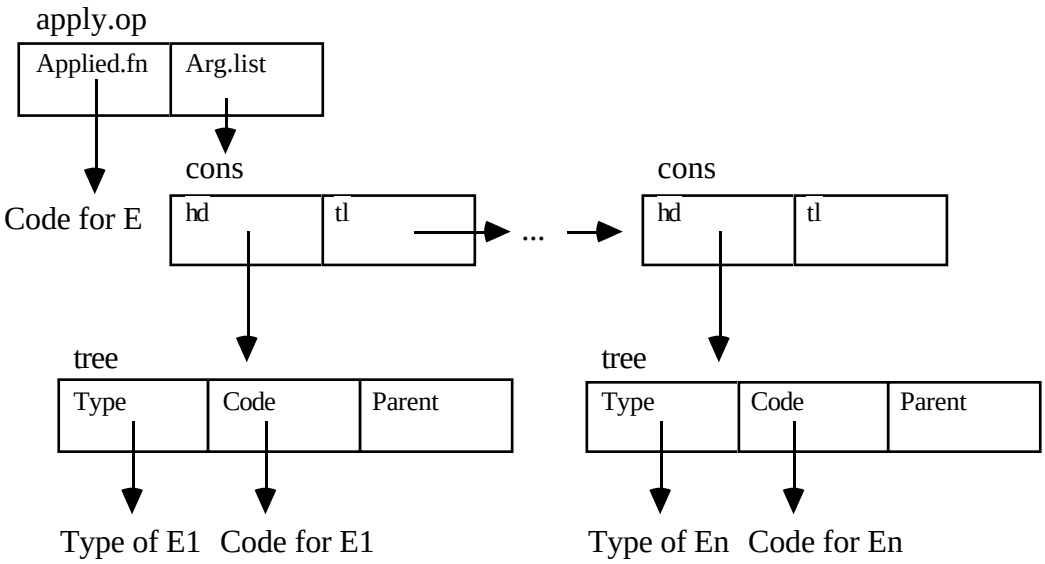
structure apply.op(pntr Apply.symbol,Arg.list)
where

Apply.symbol is a pointer to the symbol table entry for the function.
Arg.list is a cons list of the arguments.

Note that some of the language constructs which cause an apply.op to be generated are not represented as application in the syntax of PS-algol. These functions all have the special field filled in in the symbol table entry.

PS-algol source: E(E1,..,En)

PAIL code:



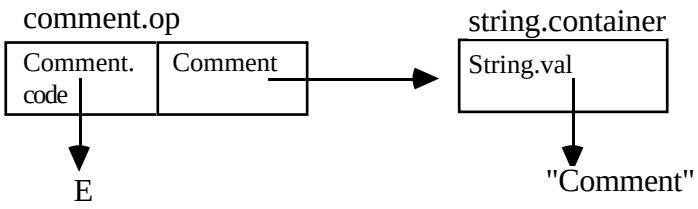
12. Comments

structure comment.op(pntr Comment.code,Comment)
where

Comment.code is a pointer to the code to which the comment pertains.
Comment is a pointer to the comment. Initially this will be a string but could be something more structured in the future (pictures etc.).

PS-algol source: ! comment
E

PAIL code:



13. Optimisations

structure optimised(pntr Optimised,Non.optimised,Optimisation.info)
where

Optimised contains the optimised code.
Non.optimised contains the source code.
Optimisation.info contains clues for the code generators.Optimisation