Item #194

FL Reference Manual

John Backus, Peter Lucas, John H. Williams, Edward L. Wimmers
IBM Research - Almaden Research Center
K53-803
650 Harry Road
San Jose, CA 95120-6099

November 23, 1988

Abstract

This is the (unreadable) reference manual for FL.

1 Review of the Domain

1. $\mathcal{I}$ is the set of non-negative integers.

2. Roughly speaking, the domain consists of the atoms and the codes for functions and is closed under sequence formation and tagging (for use with abstract data types); the elements of the set $Coll$ (consisting of $\perp$ and errors) collapse sequences and tag pairs. More formally, the set D is the smallest (free) set containing the set of atoms, the set $\{\perp\}$, and the set C closed under the following:

   (a) if $x_1, \ldots, x_n \in D - Coll$, then $\ll x_1, \ldots, x_n \gg \in D$

   (b) if $x \in D - Coll$ and $i \in \mathcal{I}$, then $\prec i, x \succ \in D$

   where the set $Coll = \{\perp\} \cup \{\prec 0, x \succ \mid x \in D - Coll\}$

3. All errors have the form $\prec 0, v \succ$ where 0 is the integer tag value for errors and v is a value. A string is a sequnce of chars. Str denotes the set of strings. System generated errors will have the form $\prec 0, \ll s_1, s_2, v \gg \succ$ where $s_1, s_2 \in Str$ and $v \in D - Coll$.

4. $OutS$ is a stream (i.e. a potentially infinite tuple) that is just to record all external events (such as I/O and file system activity). $OutS_{finite}$ is the subset of $OutS$ that corresponds to a finite number of events. More formally,
   $OutS = (D - Coll)^{\leq \omega}$ and $OutS_{finite} = (D - Coll)^{< \omega}$.

The history H consists of an input oracle (which is just an element of C) and an output stream (which is just an element of $OutS$). More formally,
$H = C \times OutS$ and $H_{finite} = C \times OutS_{finite}$.

h_{null} is the element $(nt, ()) \in H$.

$APPEND$ is a binary operator that appends the second stream onto the end of the first. (If the first already has infinite length, then the $APPEND$ operation just returns the first.)

5. An element of DH consists of an object (an element of D) and a history (an element of H) describing all the events. If there are an infinite number of events, then the computation must be non-terminating and hence the object part must be $\perp$. More formally,
$DH = (D - \{\perp\}) \times H_{finite} \cup \{\perp\} \times H$
It is also useful to have the following selector functions:

$Dof(d,(i,o)) = d$

$Iof(d,(i,o)) = i$

$Oof(d,(i,o)) = o$

$Hof(d,(i,o)) = (i,o)$

6. Even though : represents an application of a function to an argument it must be defined even when something other than a function is applied to an argument.
Formally, the infix operation : maps $D \times DH$ to DH so that
$x : (y,h) = (x,h)$ if $x \in Coll$,
$x : (y,h) = (y,h)$ if $y \in Coll$ and $x \notin Coll$,
and $x : (y,h) = (\prec 0, \ll$ "apply", "arg", $\ll x, y \gg\gg\succ, h)$ if $x \notin C \cup Coll$ and $y \notin Coll$.
: is extended to map $DH \times D$ to DH by $(x,h) : y = x : (y,h)$.
: is extended to map $D \times D$ to D by $x : y = \begin{cases} Dof(x : (y, h_{null})) & \text{if } Hof(x : (y, h_{null})) = h_{null} \\ \perp & \text{otherwise} \end{cases}$

7. A set X is *directed* iff every finite subset of X has an upper bound in X.
A partial order is *complete* iff every directed set has a least upper bound.
A complete partial order is called a *c.p.o.*
A c.p.o. is *consistently complete* iff every set with an upper bound has a least upper bound.
A function f is *continuous* iff $f(\sqcup X) = \sqcup(f(X))$ for every directed set X.

8. $\leq_d$ is a consistently complete c.p.o. on D such that

$\perp \leq_d x$ for all $x \in D$

if a is an atom, then $a \leq_d x$ iff $a = x$

if $c_1 \in C$, then $c_1 \leq_d c_2$ iff $c_2 \in C$ and $c_1 : x \leq_{dh} c_2 : x$ for all $x \in DH$.

$\ll x_1, \ldots, x_n \gg \leq_d y$ iff $y = \ll y_1, \ldots, y_n \gg$ and $x_i \leq_d y_i$ for all $i = 1, \ldots, n$.

$\prec i, x \succ \leq_d y$ iff $y = \prec i, y_0 \succ$ and $x \leq_d y_0$.

9. The ordering on DH is essentially the ordering induced by the $\leq_d$ ordering on the components. In addition, since all operations must leave the input stream unchanged, these two components must be equal. Furthermore, since a non-bottom D part indicates a completed computation, the number of events must be identical in the case that the D part is non-bottom. More formally, $\leq_{dh}$ is a consistently complete c.p.o. on DH such that $(d_1, (c_1, (x_1, \ldots, x_m))) \leq_{dh} (d_2, (c_2, (y_1, \ldots, y_n)))$ holds iff the following all hold

   $d_1 \leq_d d_2$

   $c_1 = c_2$

   $m \leq n$

   $x_j \leq_d y_j$ for all $j = 1, \ldots, m$

   either $d_1 = \bot$ or $m = n$.

   $(d_1\ h_1) \leq_{dh} (d_2\ h_2) \iff$

   $d_1 \leq_d d_2$ & $h_1 \leq_h h_2$

   $h_1 = (c_1\ (x_1\ \ x_m))$

   $h_2 = (c_2\ (y_1\ \ y_n))$

   $c_1 = c_2$

10. Let g be a function from DH to itself. Then g is

    strict iff $g(x) = x$ whenever $Dof(x) \in Coll$.

    input-preserving iff $Iof(g(x)) = Iof(x)$ for all $x \in DH$.

    output-extending iff there exists z such that $Oof(g(x)) = APPEND(Oof(x), z)$.

    history-preserving iff it is input-preserving and output-extending.

    honest iff it is continuous, strict, and history-preserving.

11. C is isomorphic to the set of honest functions from DH to itself. In particular, the function Ψ from C to the set of honest functions from DH to DH is an isomorphism where Ψ is defined by $\Psi(c)(x) = c : x$.

$(p \rightarrow f;\ id) \circ in \circ \tilde{\ } kld$

Def $f \equiv$ ~~$g \circ f \circ in \circ \tilde{\ } kld$~~

$f_0 = nt$

2 Preliminary Definitions

1. $\mathcal{N}$ is the set of function names. $\mathcal{N}^{pat}$ is the set of function names superscripted with the mark *pat*.

2. An *assignment* V is a function from $\mathcal{N} \cup \mathcal{N}^{pat}$ to elements of $D \cup \{\#\}$. Assignments are frequently denoted by the letter V. The $\#$ is used to indicate a meaningless (or non-existent) value.

3. $V_1 \leq_a V_2$ iff $V_1(\mathbf{x}) \leq_d V_2(\mathbf{x})$ for all $\mathbf{x}$ where $\#$ is viewed as strictly less than any element of D.

4. $V_1[V_2]$ represents the union of the two assignments V_1 and V_2 with V_2 winning any clash. More formally, for each $\mathbf{fn} \in \mathcal{N}$,

$$V_1[V_2](\mathbf{fn}) = \begin{cases} V_2(\mathbf{fn}) & \text{if } V_2(\mathbf{fn}) \neq \# \\ V_1(\mathbf{fn}) & \text{otherwise.} \end{cases} \qquad V_1[V_2](\mathbf{fn}^{pat}) = \begin{cases} V_2(\mathbf{fn}^{pat}) & \text{if } V_2(\mathbf{fn}) \neq \# \\ V_1(\mathbf{fn}^{pat}) & \text{otherwise.} \end{cases}$$

5. $AppDefs(V, x)$ represents applying every definition in V to the element $x \in D$. More formally, for each $\mathbf{fn} \in \mathcal{N} \cup \mathcal{N}^{pat}$ and each $x \in D$,

$$AppDefs(V, x)(\mathbf{fn}) = \begin{cases} V(\mathbf{fn}) : x & \text{if } V(\mathbf{fn}) \neq \# \\ \# & \text{otherwise.} \end{cases}$$

6. $\Lambda y.E(y) = \lambda(y, h).(E(y), h)$. Note that the in the notation $\lambda x.e$ both e and x represent elements of DH. Also note that the in the notation $\Lambda x.e$ both e and x represent elements of D.

7. PF is a total function from $\mathcal{N}$ to C. $PF(\mathbf{f})$ is the element of C that corresponds to the primitive function named by $\mathbf{f}$. If $\mathbf{f}$ does not name a primitive function,
then $PF(\mathbf{f}) = \Lambda t.(\prec 0, \ll \text{“}\mathbf{f}\text{”}, \text{“}NoSuch\text{”}, t \gg\succ)$.

f:x = apply:⟨f,x⟩

3 Some important elements of C

Of course, all functions are strict and hence if the argument to the function has the form (x,h) for some $x \in Coll$ then the function returns (x,h). If Dof applied to the argument is not in $Coll$, and if the argument fails to unify with the expression t in $\Lambda t.$, then the result is
$\prec 0, \ll f, \text{"}arg\ \ 1\text{"}, x \gg\succ$ where f is the string version of the function name and x is the argument to the function. A similar comment holds for $\Lambda t.$. The second argument to a curried function is labelled with the string "arg 2" if the second argument is erroneous.

1. $add = \Lambda \ll x_1,\ldots,x_n \gg .(\text{ sum of } 0, x_1,\ldots,x_n)$

2. $and = \Lambda \ll x_1,\ldots,x_n \gg .(\text{ Boolean and of } true, x_1,\ldots,x_n)$ where anything other than $false$ is treated as true.

3. $al = \Lambda \ll y, \ll x_1,\ldots,x_n \gg\gg .(\ll y, x_1,\ldots,x_n \gg)$

4. $apply = \lambda(\ll x_1,x_2 \gg, h).(x_1 : (x_2, h))$

5. $ar = \Lambda \ll\ll x_1,\ldots,x_n \gg, y \gg .(\ll x_1,\ldots,x_n, y \gg)$

6. $\beta = \Lambda f.\lambda(x,h).($
   $f:(x,h)$ if $Hof(f:(x,h)) = h$
   $(\bot, h)$ otherwise $)$

7. $catch = \Lambda \ll f,g \gg .\lambda(x,h).($
   $g:(\ll x,y \gg, h_1)$ if $x_1 = \prec 0, y \succ$
   (x_1,h_1) otherwise
   where $(x_1,h_1) = f:(x,h)$ $)$

8. $\circ = \Lambda \ll x_1,\ldots,x_n \gg .\lambda(y,h).(x_1 : (\ldots(x_n : (y,h))\ldots))$

9. $\circ^{pat} = \Lambda \ll x_1,\ldots,x_n \gg .(K :\ll \circ :\ll x_2,\ldots,x_n \gg,\ldots,\circ :\ll\gg\gg)$

10. $cond = \Lambda \ll x_1,x_2,x_3 \gg .\lambda(y,h).($
    (y_1,h_1) if $y_1 \in Coll$
    $x_3:(y,h_1)$ if $y_1 = false$
    $x_2:(y,h_1)$ otherwise
    where
    $(y_1,h_1) = x_1:(y,h)$ $)$

11. $cons = \Lambda \ll x_1,\ldots,x_n \gg .\lambda(y,h_0).($
    $(\ll x_1,\ldots,x_n \gg, h_1)$ if no $x_i \in Coll$
    (x_j,h_j) if $x_j \in Coll$ and no $x_i \in Coll$ for $i < j$
    where
    $(x_i,h_i) = x_i:(y,h_{i-1})$ for $i = 1,\ldots,n$ $)$

12. $Curry = \Lambda f.\Lambda x.\lambda(y,h).(f : (\ll x, y \gg, h))$

13. $div = \Lambda \ll z_1, z_2 \gg .(z_1 \text{ divided by } z_2)$

14. $FF = \Lambda x.false$

15. $\gamma = \Lambda f.\Lambda x.(f : x)$

16. $id = \Lambda x.x$

17. $IO = \lambda(x,(c,(o_1,\ldots,o_n))).(\ \ c :\ll o_1,\ldots,o_n,x \gg\ \ ,\ \ (c,(o_1,\ldots,o_n,x))\ \ )$

18. $isX = \Lambda t.(true \text{ if } t \text{ is an } x;\ false \text{ otherwise }) \text{ (for } X = atom, bool, char, func, real, seq, int)$

19. $K = \Lambda x.\Lambda y.(x)$

20. $lift = \Lambda f.(\circ :\ll f, cons \gg)$

21. $MakeBool = cond :\ll id, TT, FF \gg$

22. $mkerr = \Lambda t. \prec 0, t \succ$

23. $mul = \Lambda \ll z_1,\ldots,z_n \gg .(\text{ product of } 1, z_1,\ldots,z_n)$

24. $or = \Lambda \ll z_1,\ldots,z_n \gg .(\text{ Boolean or of } false, z_1,\ldots,z_n) \text{ where anything other than } false \text{ is treated as true.}$

25. $pcons = \Lambda \ll z_1,\ldots,z_n \gg .\lambda(y,h_0).($

$$
\begin{array}{rl}
(false, h_0) & \text{if } y \text{ does not have the form } \ll y_1,\ldots,y_n \gg \\
and : (\ll z_1,\ldots,z_n \gg, h_n) & \text{if no } z_i \in Coll \text{ where } (z_i,h_i) = x_i : (y_i, h_{i-1}) \\
 & \quad \text{and } y = \ll y_1,\ldots,y_n \gg \\
(z_j, h_j) & \text{if } z_j \in Coll \text{ and } z_i \notin Coll \text{ for } i < j \text{ where} \\
 & \quad (z_i,h_i) = x_i : (y_i, h_{i-1}) \text{ and } y = \ll y_1,\ldots,y_n \gg)
\end{array}
$$

26. $pcons^{pat} = \Lambda \ll z_1,\ldots,z_n \gg .(K :\ll s1,\ldots,sn \gg)$

27. $r1 = \Lambda \ll z_1,\ldots,z_n \gg .z_n \text{ where } n \geq 1.$

28. $s1 = \Lambda \ll z_1,\ldots,z_n \gg .z_1 \text{ where } n \geq 1.$

29. $sub = \Lambda \ll z_1, z_2 \gg .(z_2 \text{ subtracted from } z_1)$

30. $tl = \Lambda \ll z_1,\ldots,z_n \gg .(\ll z_2,\ldots,z_n \gg)$

31. $tlr = \Lambda \ll z_1,\ldots,z_n \gg .(\ll z_1,\ldots,z_{n-1} \gg)$

32. $TT = \Lambda x.true.$

Here are the primitive function names: **add**, **and**, **al**, **apply**, **ar**, β, $\circ$, **catch**, **cond**, **cons**, **Curry**, **div**, **FF**, γ, **id**, **IO**, **isatom**, **isbool**, **ischar**, **isfunc**, **isint**, **isreal**, **isseq**, **K**, **lift**, **mkerr**, **mul**, **or**, **pcons**, **rl**, **sl**, **sub**, **tl**, **tlr**, **TT**.

4 Abbreviation Elimination

Before the meaning is computed certain abbreviations are removed. In the following, $\mathbf{x}^{(n)}$ denotes the expression $\mathbf{x}$ with $\mathbf{n}$ primes on it and $\mathbf{x}^n : \mathbf{y}$ denotes $\mathbf{x} : (\ldots(\mathbf{x} : \mathbf{y})\ldots)$ where there are $\mathbf{n}$ occurences of $\mathbf{x}$ and $\overline{\mathbf{fn}}$ denotes fn **where** **Def** fn = **PF** **end** . The notation $\sqsubset \ldots \sqsupset$ is used to denote optional expressions. During abbreviation removal, the following operations are performed (in any order) until no more can be applied:

1. Enclose the expression by a where clause defining all primitive functions.
2. $[^{(n)}\mathbf{x}_1, \ldots, \mathbf{x}_n]$ is replaced by $\overline{\mathbf{lift}^n} : \overline{\mathbf{cons}} :< \mathbf{x}_1, \ldots, \mathbf{x}_n >$
3. $[\,^{(n)}\mathbf{x}_1, \ldots, \mathbf{x}_n]$ is replaced by $\overline{\mathbf{lift}^n} : \overline{\mathbf{pcons}} :< \mathbf{x}_1, \ldots, \mathbf{x}_n >$
4. $\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ is replaced by $\mathbf{x}_2 :< \mathbf{x}_1, \mathbf{x}_3 >$.
5. $\mathbf{p} \rightarrow^{(n)} \mathbf{x}$ is replaced by $\mathbf{p} \rightarrow^{(n)} \mathbf{x}; \overline{\mathbf{K}^n} : (\overline{\mathbf{mkerr}}\sigma[\text{``cond''}, \text{``''}, \overline{\mathbf{id}}])$.
6. **Def** $\mathbf{p_0} \sqsubset \mathbf{p_1} \cdots \mathbf{p_n} \sqsupset \sqsubset \leftarrow \mathbf{p} \sqsupset = \mathbf{e} \sqsubset$ **Pat** $\mathbf{E} \sqsupset$ is replaced by
   Def $\mathbf{fn} \sqsubset \mathbf{p_1} \cdots \mathbf{p_n} \sqsupset \sqsubset \leftarrow \mathbf{p} \sqsupset =$
   $(\mathbf{p_0} \rightarrow \mathbf{fn}; \overline{\mathbf{mkerr}}\sigma[\text{``fn''}, \text{``result''}, \overline{\mathbf{id}}])\sigma\mathbf{e}$
   $\sqsubset$ **Pat** $(\mathbf{p_0} \rightarrow \mathbf{fn}; \overline{\mathbf{mkerr}}\sigma[\text{``fn''}, \text{``patresult''}, \overline{\mathbf{id}}])\sigma\mathbf{E} \sqsupset$
   for each fn given a definition by the pattern $\mathbf{p_0}$.
7. **Def** $\mathbf{fn} \sqsubset (\mathbf{p_1}) \ldots (\mathbf{p_n}) \sqsupset \leftarrow \mathbf{p} = \mathbf{e} \sqsubset$ **Pat** $\mathbf{E} \sqsupset$ is replaced by
   Def $\mathbf{fn} \sqsubset (\mathbf{p_1}) \ldots (\mathbf{p_n}) \sqsupset =$
   $\overline{\mathbf{catch}} :< \mathbf{p}, \overline{\mathbf{FF}} >$ thenarme; $\overline{\mathbf{mkerr}}\sigma[\text{``fn''}, \text{``arg n + 1''}, \overline{\mathbf{id}}]$
   $\sqsubset$ **Pat** overlinecatch $:< \mathbf{p}, \overline{\mathbf{FF}} > \rightarrow \mathbf{E}; \overline{\mathbf{mkerr}}\sigma[\text{``fn''}, \text{``patarg n + 1''}, \overline{\mathbf{id}}] \sqsupset$.
8. **Def** $\mathbf{fn}(\mathbf{p_1}) \ldots (\mathbf{p_m}) = \mathbf{e} \sqsubset$ **Pat** $\mathbf{E} \sqsupset$ is replaced by
   Def $\mathbf{fn}(\mathbf{p_1}) \ldots (\mathbf{p_{m-1}}) \leftarrow \mathbf{p_m} = \lambda(\mathbf{p_m})\mathbf{e} \sqsubset$ **Pat** $\lambda(\mathbf{p_m})\mathbf{E} \sqsupset$
9. $\mathbf{x}'$ is replaced by $\overline{\mathbf{lift}} : \mathbf{x}$. $\text{``}\mathbf{x}$ is replaced by $\overline{\mathbf{K}} : \mathbf{x}$.
10. $\mathbf{f}.$ is replaced by $\mathbf{f}.\overline{\mathbf{TT}}$.
11. **let** dl **in** e is replaced by e **where** dl **end**
12. $\mathbf{x} :^{(n)} \mathbf{y}$ is replaced by $\overline{\mathbf{lift}^n} : \overline{\mathbf{apply}} :< \mathbf{x}, \mathbf{y} >$ for $\mathbf{n} \geq 1$.
13. $\mathbf{p} \rightarrow^{(n)} \mathbf{e_1}; \mathbf{e_2}$ is replaced by $\overline{\mathbf{lift}^n} : \overline{\mathbf{cond}} :< \mathbf{p}, \mathbf{e_1}, \mathbf{e_2} >$.
14. "$z_1 \ldots z_n$" is replaced by $< \text{`}z_1, \ldots, \text{`}z_n >$.

Because it is an extralingual notion and dependent on the environment in which the programmer is running FL programs, the notion of an "include" facility is not addressed in this manual. A typical preprocessor would do something like replacing the statement **include** **n** where **n** is a string by the declaration list named by **n**. Questions as to where this declaration list reside are dependent on the environment and not addressed here.

$E \to (V \to (H \to DH))$

μ

5 The Meaning of Expressions

The meaning of (a part of) an expression is given by the two (uncurried) meta-functions
$\mu : Expressions \times assignments \times H_{finite} \to DH \cup \{\#\}$
and $\rho : Expressions \times assignments \times (\mathcal{N} \cup \mathcal{N}^{pat}) \to \mathcal{D} \cup \{\#\}$.
μ and ρ are meaningful only in the case that the assignment gives a non-# meaning to every free function name in the expression.
The notation $\rho(\mathbf{e}, V, \cdot)$ is used to denote the assignment V_0 given by $V_0(\mathbf{fn}) = \rho(\mathbf{e}, V, \mathbf{fn})$.
Generally speaking, evaluation of subexpressions is done in a left-to-right fashion.

1. $\mu(\mathbf{a}, V, h) = (a, h)$ if $\mathbf{a}$ is an atom.

2. if $\mathbf{f}$ is a function name, then $\mu(\mathbf{f}, V, h) = (V(\mathbf{f}), h)$

3. $\mu(< \mathbf{x_1}, \ldots, \mathbf{x_n} >, V, h) = \begin{cases} (\ll y_1, \ldots, y_n \gg, h_n) & \text{if } \{y_1, \ldots, y_n\} \cap Coll = \emptyset \\ (y_j, h_j) & \text{if } \{y_1, \ldots, y_j\} \cap Coll = \{y_j\} \text{ and } j \leq n \end{cases}$

   where

   $h_0 = h$
   $(y_i, h_i) = \mu(\mathbf{x_i}, V, h_{i-1})$ for $i = 1, \ldots, n$

4. $\mu(\mathbf{x_1} : \mathbf{x_2}, V, h) = \begin{cases} y_1 : \mu(\mathbf{x_2}, V, h_1) & \text{if } y_1 \notin Coll \\ (y_1, h_1) & \text{if } y_1 \in Coll \end{cases}$

   where

   $(y_1, h_1) = \mu(\mathbf{x_1}, V, h)$

   $f = e$
   $e : x$

5. $\mu(\lambda(\mathbf{p})\mathbf{e}, V, h) = (\ \lambda(x, h_0).(\mu(\mathbf{e}, V[AppDefs(\rho(\mathbf{p}, V, \cdot), x)], h_0))\ ,\ h\ )$

6. $\mu(\mathbf{f}.\mathbf{x}, V, h) = \mu(\mathbf{x}, V, h)$

7. $\mu(\mathbf{xW}, V, h) = \mu(\mathbf{x}, V[\rho(\mathbf{W}, V, \cdot)], h)$ where $\mathbf{W}$ is a (generalized) where clause.

8. $\mu(\mathbf{p} \to \mathbf{x_1}; \mathbf{x_2}, V, h) = \begin{cases} (cond : \ll y_0, y_1, y_2 \gg, h_2) & \text{if } \{y_0, y_1, y_2\} \cap Coll = \emptyset \\ (y_j, h_j) & \text{if } \{y_0, \ldots, y_j\} \cap Coll = \{y_j\} \text{ and } j \leq 2 \end{cases}$

   where

   $(y_0, h_0) = \mu(\mathbf{p}, V, h)$
   $(y_1, h_1) = \mu(\mathbf{x_1}, V[\rho(\mathbf{p}, V, \cdot)], h_0)$
   $(y_2, h_2) = \mu(\mathbf{x_2}, V[\rho(\mathbf{p}, V, \cdot)], h_1)$

   $V :: \eta_{Name} \to (H \to DH)$
   $\mu :: E \to V \to H \to DH$
   $\rho :: Defs \to (V \to V)$
   $\pi :: Pats \to (V \to (V \times (H \to DH)))$

9. $\mu(\mathbf{e}, V, h) = \#$ if $\mathbf{e}$ is a (generalized) where clause or a declaration.

Peter:
$\mu(\lambda(p)e)(V)(h_1) = \lambda h_0.((\lambda(x,h).\mu(e)(V'[\{f \to apply(V_p(f), \lambda h_2.(x,h_2)) \mid f \in Dom\, V_p\}]))$ where
$V_p = \pi_1(p)(V)$

F turns an expression (together with an assignment) into a function. Specifically, $F(\mathbf{x}, V) = (\lambda(y, h).(\mu(\mathbf{x}, V, h) : y))$.

For any expression-part **e** and any assignment V, $\rho(\mathbf{e}, V, \cdot)$ is the least assignment such that the following hold for all $\mathbf{fn} \in \mathcal{N} \cup \mathcal{N}^{pat}$ and all $\mathbf{f} \in \mathcal{N}$:

1. $\rho($ **where** **rec** **dl** **end** $, V, \mathbf{fn}) = \rho(\mathbf{dl}, V[\rho(\mathbf{dl}, V, \cdot)], \mathbf{fn})$

2. $\rho($ **where** **non − rec** **dl** **end** $, V, \mathbf{fn}) = \rho(\mathbf{dl}, V, \mathbf{fn})$

3. if **t** is a definition or a hide or export clause in the definition list **dl**, then $\rho(\mathbf{t}, V, \cdot) \leq_a \rho(\mathbf{dl}, V, \cdot)$

4. $\rho(\mathbf{e}\ \ \mathbf{W}, V, \mathbf{fn}) = \rho(\mathbf{e}, V[\rho(\mathbf{W}, V, \cdot)], \mathbf{fn})$ where **W** is a (generalized) where clause.

5. $\rho($ **export** $(\mathbf{f_1}, \ldots, \mathbf{f_i})$ **dl** **end** $, V, \mathbf{fn}) = \rho((, V, \mathbf{fn})\mathbf{dl}, V[\rho(\mathbf{dl}, V, \cdot)], \cdot)$
   if $\mathbf{fn} \in \{\mathbf{f_1}, \ldots, \mathbf{f_i}, \mathbf{f_1}^{pat}, \ldots, \mathbf{f_i}^{pat}\}$

6. $\rho($ **hide** $(\mathbf{f_1}, \ldots, \mathbf{f_i})$ **dl** **end** $, V, \mathbf{fn}) = \rho(\mathbf{dl}, V[\sout{\rho(\mathbf{dl}, V, \cdot)}], \cdot)$
   if $\mathbf{fn} \notin \{\mathbf{f_1}, \ldots, \mathbf{f_i}, \mathbf{f_1}^{pat}, \ldots, \mathbf{f_i}^{pat}\}$

7. $\rho($ **Def** $\mathbf{f} =$ **PF** $, V, \mathbf{f}) = PF(\mathbf{f})$.

8. $\rho($ **Def** $\mathbf{f} = \mathbf{x}, V, \mathbf{f}) = F(\mathbf{x}, V)$

9. $\rho($ **Def** $\mathbf{f} = \mathbf{x}$ **Pat** $\mathbf{y}, V, \mathbf{f}) = F(\mathbf{x}, V)$
   $\rho($ **Def** $\mathbf{f} = \mathbf{x}$ **Pat** $\mathbf{y}, V, \mathbf{f}^{pat}) = F(\mathbf{y}, V)$

10. Let $W = \rho($ **Rep** $\mathbf{t} \Leftrightarrow \mathbf{x}, V, \cdot)$.

    $W(\mathbf{ist}) = cond :\ll istag : \mathbf{t}, \circ :\ll MakeBool, \gamma : F(\mathbf{x}, V), untag : \mathbf{t} \gg, FF \gg$
    $W(\mathbf{mkt}) = cond :\ll \gamma : F(\mathbf{x}, V), tag : \mathbf{t}, \Lambda z.(\prec 0, \ll \text{“}mkt\text{”}, \text{“}arg\ \ 1\text{”}, z \gg\succ) \gg$
    $W(\mathbf{umt}) = cond :\ll W(\mathbf{ist}), untag : \mathbf{t}, \Lambda z.(\prec 0, \ll \text{“}umt\text{”}, \text{“}arg\ \ 1\text{”}, z \gg\succ) \gg$
    $W(\mathbf{f}) = catch :\ll \circ :\ll \rho(\mathbf{x}, V, \mathbf{f}), W(\mathbf{umt}) \gg, \Lambda \ll y_1, y_2 \gg .(mkerr :\ll \text{“}f\text{”}, \text{“}arg\ \ 1\text{”}, y_1 \gg) \gg$
    if $\rho(\mathbf{x}, V, \mathbf{f}) \neq \#$ and $\mathbf{f} \neq$ **ist, mkt, umt**

11. $\rho(\mathbf{f.x}, V, \mathbf{f}) = id$
    $\rho(\mathbf{f.x}, V, \mathbf{fn}) = \rho(\mathbf{x}, V, \mathbf{fn})$ if $\mathbf{fn} \neq \mathbf{f}$.

12. If $\mathbf{i} \leq n$ and $\rho(\mathbf{y_i}, V, \mathbf{f}) \neq \#$, then $\rho(\mathbf{g} :< \mathbf{y_1}, \ldots, \mathbf{y_n} >, V, \mathbf{f}) =$

    $$\circ :\ll \quad \rho(\mathbf{y_1}, V, \mathbf{f}) \quad , \quad AppsR :\ll si, V(\mathbf{g}^{pat}), \ll F(\mathbf{y_1}, V), \ldots, F(\mathbf{y_n}, V) \gg\gg \ \gg$$

    where $AppsR = \Lambda \ll z_1, z_2, z_3 \gg .\lambda(y, h).(\ \ (z_1 : (z_2 : (z_3, h)))\ \ :\ \ y\ \ )$

6 The Meaning of Meta-Predicates

Γ is the meaning function for meta-predicates. It maps an FL (meta-)expression and an assignment to a total prediate over D.

1. $BOOL(x) = \begin{cases} FALSE & \text{if } x \in \{false\} \cup Coll \\ TRUE & \text{otherwise} \end{cases}$

2. $\Gamma(\mathbf{e}, V)(x) = \Gamma(\mathcal{P}(\mathbf{e}), V)(x) = \bigwedge\{BOOL(Dof(\mu(\mathbf{e}, V, h) : x)) | h \in H\}$ if $\mathbf{e}$ is an FL expression.

3. $\Gamma(P_1 \wedge P_2, V)(x) = \bigwedge\{\Gamma(P_1, V)(x), \Gamma(P_2, V)(x)\}$.

4. $\Gamma(P_1 \vee P_2, V)(x) = \bigvee\{\Gamma(P_1, V)(x), \Gamma(P_2, V)(x)\}$.

5. $\Gamma(P_1 \Longrightarrow P_2, V)(x) = \bigwedge\{\Gamma(P_2, V)(Dof(x : (y, h))) | \Gamma(P_1, V)(y), h \in H\}$

6. $\Gamma(P_1 \rightarrow P_2, V)(x) = \begin{cases} TRUE & \text{if } x = \ll y_1, \ldots, y_n \gg, n \geq 1, \\ & \quad \Gamma(P_1, V)(y_1), \text{ and } \Gamma(P_2, V)(\ll y_2, \ldots, y_n \gg) \\ FALSE & \text{otherwise} \end{cases}$

7. $\Gamma(P_1 \leftarrow P_2, V)(x) = \begin{cases} TRUE & \text{if } x = \ll y_1, \ldots, y_n \gg, n \geq 1, \\ & \quad \Gamma(P_1, V)(\ll y_1, \ldots, y_{n-1} \gg), \text{ and } \Gamma(P_2, V)(y_n) \\ FALSE & \text{otherwise} \end{cases}$

8. $\Gamma([\, P_1, \ldots, P_n \,], V)(x) = \begin{cases} TRUE & \text{if } x = \ll y_1, \ldots, y_n \gg, \Gamma(P_i, V)(y_i) \text{ for all } i = 1, \ldots n \\ FALSE & \text{otherwise} \end{cases}$

9. $\Gamma(seqof : P, V)(x) = \begin{cases} TRUE & \text{if } x = \ll y_1, \ldots, y_n \gg, \Gamma(P, V)(y_i) \text{ for all } i = 1, \ldots n \\ FALSE & \text{otherwise} \end{cases}$

10. $\Gamma(\forall(F_1, \ldots F_n)P(F_1, \ldots, F_n), V)(x) = \bigwedge\{\Gamma(P(P_1, \ldots, P_n), V) | P_1, \ldots, P_n$ are meta-predicates $\}$
where $P(F_1, \ldots, F_n)$ represents a meta-predicate expression with all free occurences of $F_1, \ldots, F_n$ explicitly mentioned.

The comment **Sig** $\mathbf{e} :: P$ is true iff $\Gamma(P, V)(Dof(\mu(\mathbf{e}, V, h))) = TRUE$ for all $h \in H_0$.
The comment **Asn** $\mathbf{e_1} = \mathbf{e_2}$ is true iff $\mu(\mathbf{e_1}, V, h) = \mu(\mathbf{e_2}, V, h)$ for all $h \in H_0$.
The **Sig** comments and the **Asn** comments need not be checked by an implementation for veracity. Note that if a compiler wants to make use of one of these comments, it must be checked before it can be used.