Ligo Formal Description

Syntax

The following describe the syntax of the simplify AST which is an internal of LIGO. The concrete syntax will be different depending of the choosen one but all the caterogies are present and the corresponding evaluation are the same

A LIGO program is a succession of declarations and expressions. Declarations add bindings to the environment while expressions are evaluated and yield values

$variables\ (x)$

$label\ (l)$

$constructor\ (c)$

$declaration\ (d)\ =$

$\mid type\ x\ is\ te$	(Type declaration)
$\mid const\ x\ (:\ te)?\ =\ e$	(Constant declaration)
$\mid var\ x\ (:\ te)?\ =\ e$	(Variable declaration)

$expression\ (e)\ =$

$\mid value$	(values)
$\mid built_i n$	(built-in function)
$\mid x$	(variables)
$\mid \lambda x\ .\ expr$	(lambda)
$\mid e1\ e2$	(application)
$\mid let\ x\ =\ e1\ in\ e2$	(let in)
$\mid (\ e_i\ )$	(tuple)
$\mid c\ e$	(constructor)
$\mid \{\ l_i\ =\ e_i\}$	(record)
$\mid [\ e1_i\ =\ e2_i\ ]$	(map)
$\mid [[\ e1_i\ =\ e2_i\ ]]$	(big map)
$\mid [\ e_i\ ]$	(list)
$\mid \{\ e_i\}$	(set)
$\mid e(.a_i)$	(accessor)
$\mid e1[e2]$	(look up)
$\mid match\ e\ with\ matching$	(matching)
$\mid e1;\ e2$	(sequence)
$\mid while\ e1\ do\ e2$	(loop)
$\mid x(.a_i)\ =\ e$	(assign)
$\mid SKIP$	(skip)
$\mid e\ as\ T$	(ascription)

$type\ expression\ (te)\ =$

$\mid te\ (*\ te_i)+$	(type of tuple)
$\mid (\mid l_i\ of\ te_i)$	(type of sum)
$\mid\{\ l_i\ :\ te_i\}$	(type of record)
$\mid te1\ \rightarrow\ te2$	(type of function)
$\mid l$	(type of variable)
$\mid l\ (te_i)$	(type of built in function)

$value\ (v)\ =$

$\mid literal$	(values of built-in types)
$\mid c\ v$	(values of construct types)
$\mid \lambda x\ .\ expr$	(lambda)

$literal\ =$

$\mid unit$	()
$\mid bool$	()
$\mid int$	()
$\mid nat$	()
$\mid mutez$	()
$\mid string$	()
$\mid bytes$	()
$\mid address$	()
$\mid timestamp$	()
$\mid operation$	()

$access\ (a)\ =$

$\mid int$	(for tuples)
$\mid string$	(for record)
$\mid e$	(for map)

$matching\ (m)\ =$

$\mid\{\ true\ \ =>\ \ e;\ \ false\ \ =>\ \ e;\}$	(match bool)
$\mid\{\ nil\ \ =>\ \ e;\ \ cons(hd :: tl)\ \ =>\ \ e;\}$	(match list)
$\mid\{\ none\ \ =>\ \ e;\ \ some(x)\ \ =>\ \ e;\}$	(match option)
$\mid (x_i)\ \ =>\ \ e$	(match tuple)
$\mid (const_i(x_i)\ \ =>\ \ e_i\ )$	(match variant)

$matching\ value\ (mv)\ =$

$\mid\{\ true\ \ =>\ \ v;\ \ false\ \ =>\ \ v;\}$	(match bool value)
$\mid\{\ nil\ \ =>\ \ v;\ \ cons(hd :: tl)\ \ =>\ \ v;\}$	(match list value)
$\mid\{\ none\ \ =>\ \ v;\ \ some(x)\ \ =>\ \ v;\}$	(match option value)
$\mid (x_i)\ \ =>\ \ v$	(match tuple value)
$\mid (const_i(x_i)\ \ =>\ \ v_i\ )$	(match variant value)

Evaluation of expression

The following describe how expression are evaluated to yield expressions

base

$$x \;\rightarrow\; v \; (corresponding\ value\ in\ the\ environment) \tag{ E-VARIABLE}$$

$$built\ in\ (e_i) \;\rightarrow\; built\ in\ result\ (*\ evaluated\ depending\ on\ each\ case\ *) \tag{ E-BUILTIN}$$

$$(\lambda x.e)\ v \;\rightarrow\; [\ x \;\rightarrow\; v\ ]\ e \tag{ E-LAMBDA}$$

$$\frac{e1 \;\rightarrow\; e1'}{e1\ e2 \;\rightarrow\; e1'\ e2} \tag{ E-APP1}$$

$$\frac{e2 \;\rightarrow\; e2'}{v1\ e2 \;\rightarrow\; v1\ e2'} \tag{ E-APP2}$$

$$\frac{e1 \;\rightarrow\; e1'}{let\ x = e1\ in\ e2 \;\rightarrow\; let\ x = e1'\ in\ e2} \tag{ E-LET}$$

$$let\ x = v1\ in\ e2 \;\rightarrow\; [x \rightarrow v1]\ e2 \tag{ E-LETIN}$$

$$\frac{e1 \;\rightarrow\; e1'}{e1;\ e2 \;\rightarrow\; e1';\ e2} \tag{ E-SEQ}$$

$$unit;\ e2 \;\rightarrow\; e2 \tag{ E-SEQNEXT}$$

$$\frac{e1 \;\rightarrow\; e1'}{while\ e1\ then\ e2 \;\rightarrow\; while\ e1'\ then\ e2} \tag{ E-LOOP}$$

$$while\ true (= e1)\ then\ e2 \;\rightarrow\; e2;\ while\ e1\ then\ e2 \tag{ E-LOOPTRUE}$$

$$while\ false\ then\ e2 \;\rightarrow\; unit \tag{ E-LOOPFALSE}$$

$$SKIP \;\rightarrow\; unit \tag{ E-SKIP}$$

$$\frac{e \;\rightarrow\; e'}{e\ as\ T \;\rightarrow\; e'\ as\ T} \tag{ E-ASCR1}$$

$$v\ as\ T \;\rightarrow\; v \tag{ E-ASCR2}$$

data structure

$$\frac{e \;\rightarrow\; e'}{c\ e \;\rightarrow\; c\ e'} \tag{ E-CONST}$$

$$\frac{e_j \;\rightarrow\; e_j'}{(v_i,\ e_j,\ e_k) \;\rightarrow\; (v_i,\ e_j',\ e_k)} \tag{ E-TUPLES}$$

$$\frac{e_j \;\rightarrow\; e_j'}{\{l_i = v_i,\ l_j = e_j,\ l_k = e_k\} \;\rightarrow\; \{l_i = v_i,\ l_j = e_j',\ l_k = e_k\}} \tag{ E-RECORDS}$$

$$\frac{e2_j \;\rightarrow\; e2_j'}{[e1_i = v_i,\ e1_j = e2_j,\ e1_k = e2_k] \;\rightarrow\; [e1_i = v_i,\ e1_j = e2_j',\ e1_k = e2_k]} \tag{ E-MAP}$$

$$\frac{e2_j \;\rightarrow\; e2_j'}{[[e1_i = v_i,\ e1_j = e2_j,\ e1_k = e2_k]] \;\rightarrow\; [[e1_i = v_i,\ e1_j = e2_j',\ e1_k = e2_k]]} \tag{ E-BIGMAP}$$

$$\frac{e_j \;\rightarrow\; e_j'}{[v_i,\ e_j,\ e_k] \;\rightarrow\; [v_i,\ e_j',\ e_k]} \tag{ E-LIST}$$

$$\frac{e_j \;\rightarrow\; e_j'}{\{v_i,\ e_j,\ e_k\} \;\rightarrow\; \{v_i,\ e_j',\ e_k\}} \tag{ E-SET}$$

$$\frac{e \;\rightarrow\; e'}{e(.a_i) \;\rightarrow\; e'(.a_i)} \tag{ E-ACCESS}$$

look up

$(v_i)[j] \ \rightarrow \ v_j$ (E-LUPTUPLE)

$\{l_i = v_i\}[lj] \ \rightarrow \ v_j$ (E-LUPRECORD)

$[e_i = v_i][ej] \ \rightarrow \ v_j$ (E-LUPMAP)

$[[e_i = v_i]][ej] \ \rightarrow \ v_j$ (E-LUPBIGMAP)

$[v_i][j] \ \rightarrow \ v_j$ (E-LUPLIST)

$\{v_i\}[j] \ \rightarrow \ v_j$ (E-LUPSET)

$$\frac{e \ \rightarrow \ e'}{x(.a_i) \ = \ e \ \rightarrow \ x(.a_i) \ = \ e'}$$ (E-ASSIGN)

$x(.a_i) \ = \ v \ \rightarrow \ x'(.a_i) \ with \ x' \ as \ x \ with \ field \ (.a_i) \ replace \ by \ v$ (E-ASSIGN2)

matching

$$\frac{e \ \rightarrow \ e'}{match \ e \ with \ m \ \rightarrow \ match \ e' \ with \ m}$$ (E-MATCH1)

$$\frac{m \ \rightarrow \ m'}{match \ v \ with \ m \ \rightarrow \ match \ v \ with \ m'}$$ (E-MATCH2)

$match \ v \ with \ mv \ \rightarrow \ v' \ if \ v \ => \ v' \ in \ mv$ (E-MATCH)

$$\frac{e1 \ \rightarrow \ e1'}{\{ \ true \ => \ e1; \ false \ => \ e2; \} \ \rightarrow \ \{ \ true \ => \ e1'; \ false \ => \ e2; \}}$$ (E-MAcTHBOOL1)

$$\frac{e2 \ \rightarrow \ e2'}{\{ \ true \ => \ v1; \ false \ => \ e2; \} \ \rightarrow \ \{ \ true \ => \ v1; \ false \ => \ e2'; \}}$$ (E-MAcTHBOOL2)

$$\frac{e1 \ \rightarrow \ e1'}{\{ \ nil \ => \ e1; \ cons(hd :: tl) \ => \ e2; \} \ \rightarrow \ \{ \ nil \ => \ e1'; \ cons(hd :: tl) \ => \ e2; \}}$$ (E-MATCHLIST1)

$$\frac{e2 \ \rightarrow \ e2'}{\{ \ nil \ => \ v1; \ cons(hd :: tl) \ => \ e2; \} \ \rightarrow \ \{ \ nil \ => \ v1; \ cons(hd :: tl) \ => \ e2'; \}}$$ (E-MATCHLIST2)

$$\frac{e1 \ \rightarrow \ e1'}{\{ \ none \ => \ e1; \ some(x) \ => \ e2; \} \ \rightarrow \ \{ \ none \ => \ e1'; \ some(x) \ => \ e2; \ \}}$$ (E-MATCHOPT1)

$$\frac{e2 \ \rightarrow \ e2'}{\{ \ none \ => \ v1; \ some(x) \ => \ e2; \} \ \rightarrow \ \{ \ none \ => \ v1'; \ some(x) \ => \ e2'; \ \}}$$ (E-MATCHOPT2)

$$\frac{e \ \rightarrow \ e'}{(x_i) \ => \ e \ \rightarrow \ (x_i) \ => \ e'}$$ (E-MATCHTUPLE)

$$\frac{e_j \ \rightarrow \ e'_j}{(c_i(x_i) \ => \ v_i, \ c_j(x_j) \ => \ e_j, \ c_k(x_k) \ => \ e_k) \ \rightarrow \ (c_i(x_i) \ => \ v_i, \ c_j(x_j) => e'_j, \ c_k(x_k) => e_k)}$$

(E-MATCHVARIANT)

Derive form

The following describe equivalent notation. Meaning one could be drop for the AST without change the CSTs

$e1; \ e2 \iff (\lambda x : Unit.e1) \ e2 \ with \ x \ not \ a \ free \ variable \ in \ e1$

$let \ x = e1 \ in \ e2 \iff (\lambda x : T1.e2) \ e1$