Formal Specification of the Plutus Core Language

I. Plutus Core

Plutus Core is an untyped λ calculus design to run as a transaction validation scripting language on blockchain systems. It's designed to be simple and easy to reason about using mechanized proof assistants and automated theorem provers. The grammar of the language is given in Figure 1, using an s-expression format. As is standard in λ calculi, we have variables, λ abstractions, and application. In addition to this, there are also local let bindings, data constructors which have arguments, case terms which match on term, declared names, computational primitives (success, failure, and binding), primitive values, and built-in functions. Terms live within a top-level program which consists of some declarations.

As a few examples, consider the program in Figure 2, which defines the factorial and map functions. This program is not the most readable, which is to be expected from a representation intended for machine interpretation rather than human interpretation, but it does make explicit precisely what the roles are of the various parts.

Name	n			name
Mod	l			module name
Con	c			constructor name
Int	i			int
Float	f			float
ByStr	b			bytestring
QualN	qn	$::=$	$(\texttt{qual}\ \mathrm{l}\ \mathrm{n})$	qualified name
QualC	qc	$::=$	$(\texttt{qualcon}\ \mathrm{l}\ \mathrm{c})$	qualified constructor
Tm	M	$::=$	x	variable
			$(\texttt{decname}\ qn)$	declared name
			$(\texttt{let}\ M\ x\ M)$	local declaration
			$(\texttt{lam}\ x\ M)$	λ abstraction
			$(\texttt{app}\ M\ M)$	function application
			$(\texttt{con}\ qc\ M^*)$	constructed data
			$(\texttt{case}\ M\ C^*)$	case
			$(\texttt{success}\ M)$	success
			$\texttt{failure}$	failure
			$\texttt{txhash}$	transaction hash
			$\texttt{blocknum}$	block number
			$\texttt{blocktime}$	block time
			$(\texttt{bind}\ M\ x\ M)$	computation bind
			$(\texttt{primInt}\ i)$	primitive int
			$(\texttt{primFloat}\ f)$	primitive float
			$(\texttt{primByteString}\ b)$	primitive bytestring
			$(\texttt{builtin}\ \mathrm{n}\ M^*)$	built-in function
			$(\texttt{isFun}\ M)$	function test
			$(\texttt{isCon}\ M)$	data test
			$(\texttt{isConName}\ qc\ M)$	named data test
			$(\texttt{isInt}\ M)$	int test
			$(\texttt{isFloat}\ M)$	float test
			$(\texttt{isByteString}\ M)$	bytestring test
Cl	C	$::=$	$(\texttt{cl}\ qc\ (x^*)\ M)$	case clause
Prg	G	$::=$	$(\texttt{program}\ L^*)$	program
Mod	L	$::=$	$(\texttt{module}\ \mathrm{n}\ D^*)$	module
Dec	D	$::=$	$(\texttt{exp}\ \mathrm{n}\ M)$	exported name decl.
			$(\texttt{loc}\ \mathrm{n}\ M)$	local name decl.
			$(\texttt{expcon}\ c)$	exported constructor
			$(\texttt{loccon}\ c)$	local constructor

Fig. 1. Grammar of Plutus Core

II. Scope Correctness

We define for Plutus Core a scope correctness judgment, which explains when a program's use of scoped objects (variables and names) is valid. This judgment is defined inductively as shown in Figure 4, with auxiliary judgments in Figure 6.

Plutus Core's module system defines two kinds of declared names. Names declared using the `exp` and `expcon` keywords are exported declaration of terms and constructors, respectively, and are usable by any other module. Names declared using the `loc` and `loccon` keywords are local, and usable only in their declaring modules. This distinction between exported and local names permits a simple form of abstraction at the module level.

III. Execution and Reduction

The execution of a program in Plutus Core does not in itself result in any reduction. Instead, the declarations are bound to their appropriate names in a declaration environment δ, which we will represent by a list of items of the form $\mathrm{n} \mapsto M$. Then, designated names can be chosen to be reduced in this declaration environment generated. For instance, we might designate the name main to be the name who's definition we reduce, as is done in Haskell. The generation judgment for generating this is given by the relation $P \ggg \delta$, defined in Figure 7.

To give the computation rules for Plutus Core, we must define what the values are of the language, as given in Figure 8. Rather than using values directly, we wrap them in a return value form, because reduction steps can fail. These then let us define a parameterized binary relation $M \rightarrow^*_\delta R$ which means M reduces to R using declarations δ, in Figure 11. This uses a standard contextual dynamics to separate the local reductions, reduction contexts, and repeated reductions into separate judgments. We also define a step-indexed dynamics $M \rightarrow^n_\delta R$, which means that M reduces to R using δ in at most n steps. Step-indexed reduction is useful in settings where we want to limit the number of computational steps that

```
(program
  (module Ex
    (exp fibonacci
      (lam n
        (case (builtin equalsInt n (prim 0))
          (cl (qualcon Ex True) ()
            (prim 1))
          (cl (qualcon Ex False) ()
            (builtin multiplyInt
              n
              (app
                (decname (qual Ex fibonacci))
                (builtin subtractInt n (prim 1)))))))))
    (exp map
      (lam f
        (lam xs
          (case xs
            (cl (qualcon Ex Nil) ()
              (con (qualcon Ex Nil)))
            (cl (qualcon Ex Cons) (x xs')
              (con (qualcon Ex Cons)
                (app f x)
                (app (app (decname (qual Ex map))
                          f)
                     xs'))))))))))
```

Fig. 2. Example with Fibonacci and Map

NomCtx	Δ	$::=$	ϵ	empty nominal context
			Δ, ν	non-empty nominal context
Nom	ν	$::=$	$\mathrm{l}\ \mathit{loc}\ \mathrm{n}$	local name
			$\mathrm{l}\ \mathit{exp}\ \mathrm{n}$	exported name
			$\mathrm{l}\ \mathit{loccon}\ \mathrm{c}$	local constructor
			$\mathrm{l}\ \mathit{expcon}\ \mathrm{c}$	exported constructor
VarCtx	Γ	$::=$	ϵ	empty variable context
			Γ, x	non-empty variable context
ModCtx	Λ	$::=$	ϵ	empty module context
			Λ, l	non-empty module context

Fig. 3. Contexts

can occur.

Note that while case clauses can in principle be redundant, with the same constructor used in multiple clauses of the case expression, the definition of matching renders that redundancy irrelevant. The first clause with a matching constructor is the only one that is ever used.

Note that the success and failure terms are not effectful. That is to say, `failure` does not throw an exception of any sort. They are merely primitive values that represent computational success and failure. They are analogous to Haskell Maybe values, except that they cannot be inspected,

$\boxed{\Delta\ ;\ \Gamma\ \vdash\ M\ \mathit{term}\ \mathrm{l}}$

Term M in module l is well-formed with names in nominal context Δ and variables in variable context Γ

$$\frac{\Gamma\ \ni\ x}{\Delta\ ;\ \Gamma\ \vdash\ x\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ \mathit{permits}\ qn\ \mathit{in}\ l}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{decname}\ qn)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M_0\ \mathit{term}\ \mathrm{l} \qquad \Delta\ ;\ \Gamma, x\ \vdash\ M_1\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{let}\ M_0\ x\ M_1)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma, x\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{lam}\ x\ M)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M_0\ \mathit{term}\ \mathrm{l} \qquad \Delta\ ;\ \Gamma\ \vdash\ M_1\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{app}\ M_0\ M_1)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ \mathit{permits\ constructor}\ qc\ \mathit{in}\ l \qquad \Delta\ ;\ \Gamma\ \vdash\ M_i\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{con}\ qc\ \vec{M})\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M\ \mathit{term}\ \mathrm{l} \qquad \Delta\ ;\ \Gamma\ \vdash\ C_i\ \mathit{clause}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{case}\ M\ \vec{C})\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{success}\ M)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{}{\Delta\ ;\ \Gamma\ \vdash\ \texttt{failure}\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M_0\ \mathit{term}\ \mathrm{l} \qquad \Delta\ ;\ \Gamma, x\ \vdash\ M_1\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{bind}\ M_0\ x\ M_1)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{}{\Delta\ ;\ \Gamma\ \vdash\ \texttt{txhash}\ \mathit{term}\ \mathrm{l}}$$

$$\frac{}{\Delta\ ;\ \Gamma\ \vdash\ \texttt{blocknum}\ \mathit{term}\ \mathrm{l}}$$

$$\frac{}{\Delta\ ;\ \Gamma\ \vdash\ \texttt{blocktime}\ \mathit{term}\ \mathrm{l}}$$

Fig. 4. Scope Correctness

and all computational control is done via the *bind* construct.

IV. Basic Validation Program Structure

The basic way that validation is done in Plutus Core is slightly different than in Bitcoin Script. Whereas in Bitcoin Script, a validation is successful if the validating script successfully executes and leads *true* on the top of the stack, in Plutus Core, we have special data constructs for validation. In particular, the (`success` V) and `failure`. Any program which validates a transaction must declare a function (`decname` (`qual` Validator validator)), while the corresponding program supplied by the redeemer must declare (`decname` (`qual` Redeemer redeemer)). The declarations of both are combined into a single set of declarations, and these two declared terms are then composed with a bind. The overall validation, therefore, involves reducing the term

$$\overline{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{primInt}\ i)\ \mathit{term}\ \mathrm{l}}$$

$$\overline{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{primFloat}\ f)\ \mathit{term}\ \mathrm{l}}$$

$$\overline{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{primByteString}\ b)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M_i\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{builtin}\ \mathrm{n}\ \vec{M})\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{isFun}\ M)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{isCon}\ M)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{isConName}\ n\ M)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{isInt}\ M)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{isFloat}\ M)\ \mathit{term}\ \mathrm{l}}$$

$$\frac{\Delta\ ;\ \Gamma\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{isByteString}\ M)\ \mathit{term}\ \mathrm{l}}$$

Fig. 5. Scope Correctness (cont.)

$$
\begin{array}{l}
(\texttt{bind} \\
\quad (\texttt{decname}\ (\texttt{qual}\ \text{Redeemer redeemer})) \\
\quad x \\
\quad (\texttt{app}\ (\texttt{decname}\ (\texttt{qual}\ \text{Validator validator}))\ x))
\end{array}
$$

If this reduces to $(\texttt{ok}\ (\texttt{success}\ V))$ for some V, then the transaction is valid, analogous to Bitcoin Script successfully executing and leaving $true$ on the top of stack. On the other hand, if it reduces to $(\texttt{ok failure})$ or to $\texttt{err}$, then the transaction is invalid, analogous to Bitcoin Script either leaving $false$ on the top of stack, or failing to execute. The value returned in the success case is irrelevant to validation but may be used for other purposes.

$\boxed{\Delta\ \mathit{permits}\ qn\ \mathit{in}\ l}$

Nominal context Δ permits the use of qualified name qn in module l

$$\frac{\Delta\ \ni\ \mathrm{l}'\ \mathit{exp}\ \mathrm{n}}{\Delta\ \mathit{permits}\ (\texttt{qual}\ \mathrm{l}'\ \mathrm{n})\ \mathit{in}\ l}$$

$$\frac{\Delta\ \ni\ \mathrm{l}'\ \mathit{loc}\ \mathrm{n} \qquad l = l'}{\Delta\ \mathit{permits}\ (\texttt{qual}\ \mathrm{l}'\ \mathrm{n})\ \mathit{in}\ l}$$

$\boxed{\Delta\ \mathit{permits\ constructor}\ qc\ \mathit{in}\ l}$

Nominal context Δ permits the use of qualified constructor name qc in module l

$$\frac{\Delta\ \ni\ \mathrm{l}'\ \mathit{expcon}\ \mathrm{c}}{\Delta\ \mathit{permits\ constructor}\ (\texttt{qualcon}\ \mathrm{l}'\ \mathrm{c})\ \mathit{in}\ l}$$

$$\frac{\Delta\ \ni\ \mathrm{l}'\ \mathit{loccon}\ \mathrm{c} \qquad l = l'}{\Delta\ \mathit{permits\ constructor}\ (\texttt{qualcon}\ \mathrm{l}'\ \mathrm{c})\ \mathit{in}\ l}$$

$\boxed{\Delta\ ;\ \Gamma\ \vdash\ C\ \mathit{clause}\ \mathrm{l}}$

Clause C in module l is well-formed with names in nominal context Δ and variables in variable context Γ

$$\frac{\Delta\ \mathit{permits\ constructor}\ qc\ \mathit{in}\ l \qquad \Delta\ ;\ \Gamma, \vec{x}\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ ;\ \Gamma\ \vdash\ (\texttt{cl}\ qc\ (\vec{x})\ M)\ \mathit{clause}\ \mathrm{l}}$$

$\boxed{\Delta\ \vdash\ D\ \mathit{decl}\ \mathrm{l}\ \dashv\ \Delta'}$

Declaration D in module l is well-formed with names in nominal context Δ, producing new nominal context Δ'

$$\frac{\Delta \not\ni \mathrm{l}\ \mathit{exp}\ \mathrm{n} \qquad \Delta \not\ni \mathrm{l}\ \mathit{loc}\ \mathrm{n} \qquad \Delta\ ;\ \epsilon\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ \vdash\ (\texttt{loc}\ \mathrm{n}\ M)\ \mathit{decl}\ \mathrm{l}\ \dashv\ \Delta, \mathrm{l}\ \mathit{loc}\ \mathrm{n}}$$

$$\frac{\Delta \not\ni \mathrm{l}\ \mathit{exp}\ \mathrm{n} \qquad \Delta \not\ni \mathrm{l}\ \mathit{loc}\ \mathrm{n} \qquad \Delta\ ;\ \epsilon\ \vdash\ M\ \mathit{term}\ \mathrm{l}}{\Delta\ \vdash\ (\texttt{exp}\ \mathrm{n}\ M)\ \mathit{decl}\ \mathrm{l}\ \dashv\ \Delta, \mathrm{l}\ \mathit{exp}\ \mathrm{n}}$$

$$\frac{\Delta \not\ni \mathrm{l}\ \mathit{expcon}\ \mathrm{c} \qquad \Delta \not\ni \mathrm{l}\ \mathit{loccon}\ \mathrm{c}}{\Delta\ \vdash\ (\texttt{loccon}\ n)\ \mathit{decl}\ \mathrm{l}\ \dashv\ \Delta, \mathrm{l}\ \mathit{loccon}\ \mathrm{c}}$$

$$\frac{\Delta \not\ni \mathrm{l}\ \mathit{expcon}\ \mathrm{c} \qquad \Delta \not\ni \mathrm{l}\ \mathit{loccon}\ \mathrm{c}}{\Delta\ \vdash\ (\texttt{expcon}\ n)\ \mathit{decl}\ \mathrm{l}\ \dashv\ \Delta, \mathrm{l}\ \mathit{expcon}\ \mathrm{c}}$$

$\boxed{\Lambda\ ;\ \Delta\ \vdash\ L\ \mathit{module}\ \dashv \Lambda'\ ;\ \Delta'}$

Module L is well-formed in module context Λ with names in nominal context Δ, producing new module context Λ' and new nominal context Δ'

$$\frac{\Lambda\ \not\ni\ \mathrm{l} \qquad \Delta_i\ \vdash\ D_i\ \mathit{decl}\ \mathrm{l}\ \dashv\ \Delta_{i+1}}{\Lambda\ ;\ \Delta_0\ \vdash\ (\texttt{module}\ \mathrm{l}\ \vec{D})\ \mathit{module}\ \dashv \Lambda, \mathrm{l}\ ;\ \Delta_n}$$

$\boxed{\Delta\ \vdash\ G\ \mathit{program}\ \dashv \Delta'}$

Program G is well-formed in nominal context Δ, producing new nominal context Δ'

$$\frac{\Lambda_i\ ;\ \Delta_i\ \vdash\ L_i\ \mathit{module}\ \dashv \Lambda_{i+1}\ ;\ \Delta_{i+1}}{\Delta_0\ \vdash\ (\texttt{program}\ \vec{L})\ \mathit{program}\ \dashv \Delta_n}$$

Fig. 6. Auxiliary Scope Correctness Judgments

$\boxed{P \ \ggg \ \delta}$

Program P generates environment δ

$$\frac{M_i \ \gg \ \delta_i}{(\texttt{program}\ \vec{M}) \ \ggg \ \vec{\delta}}$$

$\boxed{M \ \gg \ \delta}$

Module M generates environment δ

$$\frac{}{(\texttt{module}\ \mathrm{l}\ \overrightarrow{(\texttt{exp}\ \mathrm{n}\ M)}) \ \gg \ \mathrm{n}_i \mapsto M_i}$$

Fig. 7. Declaration Environment Generation

$$\begin{array}{llll}
\text{Val} & V & ::= & (\texttt{lam}\ x\ M) & \lambda \text{ abstraction} \\
& & & (\texttt{con}\ n\ V^*) & \text{constructed data} \\
& & & (\texttt{success}\ V) & \text{success} \\
& & & \texttt{failure} & \text{failure} \\
& & & \texttt{txhash} & \text{transaction hash} \\
& & & \texttt{blocknum} & \text{block number} \\
& & & \texttt{blocktime} & \text{block time} \\
& & & (\texttt{primInt}\ i) & \text{int value} \\
& & & (\texttt{primFloat}\ f) & \text{float value} \\
& & & (\texttt{primByteString}\ b) & \text{bytestring value} \\
\text{Ret} & R & ::= & (\texttt{ok}\ M) & \text{returned value} \\
& & & \texttt{err} & \text{error}
\end{array}$$

Fig. 8. Values of Plutus Core

$$\begin{array}{llll}
\text{Ctx} & K & ::= & \circ & \text{hole} \\
& & & (\texttt{let}\ K\ x\ M) & \text{let context} \\
& & & (\texttt{app}\ K\ M) & \text{left app context} \\
& & & (\texttt{app}\ V\ K) & \text{right app context} \\
& & & (\texttt{con}\ qc\ V^*\ K\ M^*) & \text{condata context} \\
& & & (\texttt{case}\ K\ C^*) & \text{case context} \\
& & & (\texttt{success}\ K) & \text{success context} \\
& & & (\texttt{bind}\ K\ x\ M) & \text{bind context} \\
& & & (\texttt{builtin}\ \mathrm{n}\ V^*\ K\ M^*) & \text{builtin context} \\
& & & (\texttt{isFun}\ K) & \text{function test context} \\
& & & (\texttt{isCon}\ K) & \text{data test context} \\
& & & (\texttt{isConName}\ qn\ K) & \text{named data text context} \\
& & & (\texttt{isInt}\ K) & \text{int test context} \\
& & & (\texttt{isFloat}\ K) & \text{float test context} \\
& & & (\texttt{isByteString}\ K) & \text{bytestring test context}
\end{array}$$

Fig. 9. Grammar of Reduction Contexts

$$\begin{aligned}
\circ\{N\} &= N \\
(\texttt{let}\ K\ x\ M)\{N\} &= (\texttt{let}\ K\{N\}\ x\ M) \\
(\texttt{app}\ K\ M)\{N\} &= (\texttt{app}\ K\{N\}\ M) \\
(\texttt{app}\ M\ K)\{N\} &= (\texttt{app}\ M\ K\{N\}) \\
(\texttt{con}\ qc\ \vec{V}\ K\ \vec{M})\{N\} &= (\texttt{con}\ qc\ \vec{V}\ K\{N\}\ \vec{M}) \\
(\texttt{case}\ K\ \vec{C})\{N\} &= (\texttt{case}\ K\{N\}\ \vec{C}) \\
(\texttt{success}\ K)\{N\} &= (\texttt{success}\ K\{N\}) \\
(\texttt{bind}\ K\ x\ M)\{N\} &= (\texttt{bind}\ K\{N\}\ x\ M) \\
(\texttt{isFun}\ K)\{N\} &= (\texttt{isFun}\ K\{N\}) \\
(\texttt{isCon}\ K)\{N\} &= (\texttt{isCon}\ K\{N\}) \\
(\texttt{isConName}\ qc\ K)\{N\} &= (\texttt{isConName}\ qc\ K\{N\}) \\
(\texttt{isInt}\ K)\{N\} &= (\texttt{isInt}\ K\{N\}) \\
(\texttt{isFloat}\ K)\{N\} &= (\texttt{isFloat}\ K\{N\}) \\
(\texttt{isByteString}\ K)\{N\} &= (\texttt{isByteString}\ K\{N\})
\end{aligned}$$

Fig. 10. Context Insertion

$\boxed{M \;\; \rightarrow^{*}_{h,bn,t,\delta} \;\; R}$

Term M reduces in some number of steps to return value R in declaration environment δ, using transaction hash h, block number bn, and block time t

$$\frac{}{V \;\; \rightarrow^{*}_{h,bn,t,\delta} \;\; (\texttt{ok}\ V)}$$

$$\frac{M \;\; \rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ M') \qquad M' \;\; \rightarrow^{*}_{h,bn,t,\delta} \;\; R}{M \;\; \rightarrow^{*}_{h,bn,t,\delta} \;\; R}$$

$$\frac{M \;\; \rightarrow_{h,bn,t,\delta} \;\; \texttt{err}}{M \;\; \rightarrow^{*}_{h,bn,t,\delta} \;\; \texttt{err}}$$

$\boxed{M \;\; \rightarrow^{n}_{h,bn,t,\delta} \;\; R}$

Term M reduces in at most n steps to return value R in declaration environment δ, using transaction hash h, block number bn, and block time t

$$\frac{}{V \;\; \rightarrow^{n}_{h,bn,t,\delta} \;\; (\texttt{ok}\ V)}$$

$$\frac{M \;\; \rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ M')}{M \;\; \rightarrow^{0}_{h,bn,t,\delta} \;\; \texttt{err}}$$

$$\frac{M \;\; \rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ M') \qquad M' \;\; \rightarrow^{n}_{h,bn,t,\delta} \;\; R}{M \;\; \rightarrow^{n+1}_{h,bn,t,\delta} \;\; R}$$

$$\frac{M \;\; \rightarrow_{h,bn,t,\delta} \;\; \texttt{err}}{M \;\; \rightarrow^{n}_{h,bn,t,\delta} \;\; \texttt{err}}$$

$\boxed{M \;\; \rightarrow_{h,bn,t,\delta} \;\; R}$

Term M reduces in one step to return value R in declaration environment δ, using transaction hash h, block number bn, and block time t

$$\frac{M \;\; \Rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ M')}{K\{M\} \;\; \rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ K\{M'\})}$$

$$\frac{M \;\; \Rightarrow_{h,bn,t,\delta} \;\; \texttt{err}}{K\{M\} \;\; \rightarrow_{h,bn,t,\delta} \;\; \texttt{err}}$$

Fig. 11. Reduction via Contextual Dynamics

$\boxed{M \;\; \Rightarrow_{h,bn,t,\delta} \;\; R}$

Term M locally reduces to return value R in declaration context δ, using transaction hash h, block number bn, and block time t

$$\frac{}{(\texttt{decname}\ qn) \;\; \Rightarrow_{h,bn,t,\delta,\mathrm{qn}\mapsto M} \;\; (\texttt{ok}\ M)}$$

$$\frac{V \;\; = \;\; (\texttt{lam}\ x\ M')}{(\texttt{app}\ V\ V') \;\; \Rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ [V'/x]M')}$$

$$\frac{V \;\; \neq \;\; (\texttt{lam}\ x\ M')}{(\texttt{app}\ V\ V') \;\; \Rightarrow_{h,bn,t,\delta} \;\; \texttt{err}}$$

$$\frac{V \;\; = \;\; (\texttt{con}\ qc\ \vec{V'}) \qquad qc\ ,\ \vec{V'} \;\sim\; \vec{C} \;\triangleright\; R}{(\texttt{case}\ V\ \vec{C}) \;\; \Rightarrow_{h,bn,t,\delta} \;\; R}$$

$$\frac{V \;\; \neq \;\; (\texttt{con}\ qc\ \vec{V})}{(\texttt{case}\ V\ \vec{C}) \;\; \Rightarrow_{h,bn,t,\delta} \;\; \texttt{err}}$$

$$\frac{V \;\; = \;\; \texttt{failure}}{(\texttt{bind}\ V\ x\ M) \;\; \Rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ \texttt{failure})}$$

$$\frac{V \;\; = \;\; \texttt{txhash}}{(\texttt{bind}\ V\ x\ M) \;\; \Rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ [(\texttt{primByteString}\ h)/x]M)}$$

$$\frac{V \;\; = \;\; \texttt{blocknum}}{(\texttt{bind}\ V\ x\ M) \;\; \Rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ [(\texttt{primInt}\ bn)/x]M)}$$

$$\frac{V \;\; = \;\; \texttt{blocktime}}{(\texttt{bind}\ V\ x\ M) \;\; \Rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ [(\texttt{primByteString}\ t)/x]M)}$$

$$\frac{V \;\; = \;\; (\texttt{success}\ V')}{(\texttt{bind}\ V\ x\ M) \;\; \Rightarrow_{h,bn,t,\delta} \;\; (\texttt{ok}\ [V'/x]M)}$$

$$\frac{V \;\; \neq \;\; \texttt{failure} \qquad V \;\; \neq \;\; (\texttt{success}\ V')}{(\texttt{bind}\ V\ x\ M) \;\; \Rightarrow_{h,bn,t,\delta} \;\; \texttt{err}}$$

$$\frac{\text{builtin } n \text{ reduces on } \vec{V} \text{ to } R}{(\texttt{builtin}\ \mathrm{n}\ \vec{V}) \;\; \Rightarrow_{h,bn,t,\delta} \;\; R}$$

Fig. 12. Local Reduction

$$\frac{V \;=\; (\texttt{lam}\ x\ M)}{(\texttt{isFun}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim True})))}$$

$$\frac{V \;\neq\; (\texttt{lam}\ x\ M)}{(\texttt{isFun}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim False})))}$$

$$\frac{V \;=\; (\texttt{con}\ qc\ \vec{V'})}{(\texttt{isCon}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim True})))}$$

$$\frac{V \;\neq\; (\texttt{con}\ qc\ \vec{V'})}{(\texttt{isCon}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim False})))}$$

$$\frac{V \;=\; (\texttt{con}\ qc\ \vec{V'})}{(\texttt{isConName}\ qc\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim True})))}$$

$$\frac{V \;\neq\; (\texttt{con}\ qc\ \vec{V'})}{(\texttt{isConName}\ qc\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim False})))}$$

$$\frac{V \;=\; (\texttt{primInt}\ i)}{(\texttt{isInt}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim True})))}$$

$$\frac{V \;\neq\; (\texttt{primInt}\ i)}{(\texttt{isInt}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim False})))}$$

$$\frac{V \;=\; (\texttt{primFloat}\ f)}{(\texttt{isFloat}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim True})))}$$

$$\frac{V \;\neq\; (\texttt{primFloat}\ f)}{(\texttt{isFloat}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim False})))}$$

$$\frac{V \;=\; (\texttt{primByteString}\ b)}{(\texttt{isByteString}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim True})))}$$

$$\frac{V \;\neq\; (\texttt{primByteString}\ b)}{(\texttt{isByteString}\ V) \ \Rightarrow_{h,bn,t,\delta}\ (\texttt{ok}\ (\texttt{con}\ (\texttt{qualcon}\ \text{Prim False})))}$$

Fig. 13. Local Reduction (cont.)

$$\boxed{qc\ ,\ \vec{V}\ \sim\ \vec{C}\ \triangleright\ R}$$

Constructor qc with arguments $\vec{V}$ matches clauses $\vec{C}$ to produce result R

$$\frac{}{qc\ ,\ \vec{V}\ \sim\ \epsilon\ \triangleright\ \texttt{err}}$$

$$\frac{qc \;=\; qc' \qquad |\vec{V}| \;=\; |\vec{x}|}{qc\ ,\ \vec{V}\ \sim\ (\texttt{cl}\ qc'\ (\vec{x})\ M), \vec{C}\ \triangleright\ (\texttt{ok}\ [\vec{V'}/\vec{x}]M)}$$

$$\frac{qc \;=\; qc' \qquad |\vec{V}| \;\neq\; |\vec{x}|}{qc\ ,\ \vec{V}\ \sim\ (\texttt{cl}\ qc'\ (\vec{x})\ M), \vec{C}\ \triangleright\ \texttt{err}}$$

$$\frac{qc \;\neq\; qc' \qquad qc\ ,\ \vec{V}\ \sim\ \vec{C}\ \triangleright\ R}{qc\ ,\ \vec{V}\ \sim\ (\texttt{cl}\ qc'\ (\vec{x})\ M), \vec{C}\ \triangleright\ R}$$

Fig. 14. Case Matching