Scilla

LANGUAGE GRAMMAR

Ilya Sergey

ilyasergey.net

Types

Primitive type	P	::=	`Int`	Integer
			`String`	String
			`Hash`	Hash
			`BNum`	Block number
			`Address`	Account address
Type	T, S	::=	P	primitive type
			`Map` $P\ T$	map
			`Message`	message
			T `->` S	value function
			$\mathcal{D}\ \langle T_k \rangle$	instantiated data type
			α	type variable
			`forall` $\alpha . T$	polymorphic function

Expressions (pure)

Expression	e	::=	f	simple expression
			`let` x $\langle$: $T\rangle$ = f `in` e	let-form
Simple expression	f	::=	l	primitive literal
			x	variable
			{ $\langle entry\rangle_k$ }	Message
			`fun` (x : T) => e	function
			`builtin` b $\langle x_k\rangle$	built-in application
			x $\langle x_k\rangle$	application
			`tfun` α => e	type function
			@x T	type instantiation
			C $\langle$ {$\langle T_k\rangle$} $\rangle$ $\langle x_k\rangle$	constructor instantiation
			`match` x `with` $\langle$ | sel_k $\rangle$ `end`	pattern matching
Selector	sel	::=	pat => e	
Pattern	pat	::=	x	variable binding
			C $\langle pat_k\rangle$	constructor pattern
			(pat)	paranthesized pattern
			_	wildcard pattern
Message entrry	$entry$	::=	$b : x$	
Name	b			identifier

Statements (effectful)

`s ::=`	`x <- f`	read from mutable field
	`f := x`	store to a field
	`x = e`	assign a pure expression
	`match` `x` **`with`** `⟨pat => s⟩` **`end`**	pattern matching and branching
	`x <- &B`	read from blockchain state
	`accept`	accept incoming payment
	`send` `ms`	send list of messages