OMaskl

Math expressions:

Expression	Semantics
i	Integer literal.
f	Float literal.
v	Math variable, which is evaluated by finding the binding in the environment.
e1 binop e2	Math infix binary operation, which evaluates to v if e1 evaluates to math value v1 and e2 evaluates to math value v2 and v1 binop v2 is v.
binop e1 e2	Math prefix binary operation, which evaluates to v if e1 evaluates to math value v1 and e2 evaluates to math value v2 and binop v1 v2 is v.
unop e	Math unary operation, which evaluates to v if e evaluates to math value v1 and unop v1 is v.

Math operators:

Operator	Semantics
+_+	Infix math addition binary operator
-_-	Infix math subtraction binary operator
/_/	Infix math division binary operator
_	Infix math multiplication binary operator
%_%	Infix math mod binary operator
^_^	Infix math power binary operator
sqrt	Prefix math square root unary operator
abs	Prefix math absolute value unary operator
min	Prefix math minimum binary operator

max	Prefix math maximum binary operator
floor	Prefix math floor unary operator
ceil	Prefix math ceiling unary operator

String expressions:

Expression	Semantics
s	String literal
v	String variable, which is evaluated by finding the binding in the environment
e1 $_$ e2	String concatenation, which evaluates to v if e1 evaluates to string v1, e2 evaluates to string v2 and v1 $_$ v2 is v

Boolean expressions:

Expression	Semantics
b	Boolean literal
v	Boolean variable, which is evaluated by finding the binding in the environment
!_! e	Boolean negation, which evaluates to true if e evaluates to false and vice versa.
e1 binop e2	Boolean infix binary operation, which evaluates to v if e1 evaluates to boolean value v1 and e2 evaluates to boolean value v2 and v1 binop v2 is v.
e1 comp e2	Boolean infix comparison operation, which evaluates to v if e1 evaluates to math value v1 and e2 evaluates to math value v2 and v1 comp v2 is v.

Boolean Logic and Operators

Operator	Semantics

&_&	Infix logical AND binary operator
|_|	Infix logical OR binary operator
!_|	Infix logical NOR binary operator
!_&	Infix logical NAND binary operator
&_|	Infix logical XOR binary operator
=_=	Infix equality binary operator, which is only defined for math values
>_>	Infix “greater” binary operator, which is only defined for math values
<_<	Infix “less than” binary operator, which is only defined for math values
>=_=>	Infix “greater than or equal to” binary operator, which is only defined for math values
<=_=<	Infix “less than or equal to” binary operator, which is only defined for math values
!=_=!	Infix Inequality binary operator, which is only defined for math values

Conditional Expressions:

Expression	**Semantics**
if {e1} then e2 else e3	Conditional expression, which evaluates to v if e1 evaluates to true and e2 evaluates to v or if e1 evaluates to false and e3 evaluates to v. Requires that e2, e3 evaluate to values of the same type.

Variable Definitions and Expressions:

Syntax	**Effect**
T var := e	Variable definition, which evaluates to environment env which is the current environment extended to bind v to var if e evaluates to v.

T var := e1 in e2	Variable expression, which evaluates to v if e2 evaluates to v in the environment extended with the binding of v1 to var if e1 evaluates to v1.
var	Variable

Function Expressions:

Syntax	**Semantics**
func<T> f := e1 in e2	Function expression, which evaluates to v if e2 evaluates to v in the environment extended with the binding of v1 to f if e1 evaluates to function closure v1. Allows recursion.
func<T> f := anon	Function definition, which evaluates to a new environment env by extending the current environment to include f bound to cl if anon evaluates to closure cl under the current environment. Allows recursion.
[T1 var1,..., Tn varn] ->_-> {e}	Anonymous function, which evaluates to function closure cl([var1,...,varn], e, env) containing the list of variables, body, and current environment.
[T1 var1,..., Tn varn] ->_-> {e} [a1,..., an]	Anonymous function application, which evaluates to v if the anonymous function evaluates to function closure cl([var1,...,varn], e, env), [a1,...,an] evaluates to [v1,...,vn], and evaluating e in env extended with the bindings obtained by binding [var1,...,varn] to [v1,...,vn] evaluates to v.
name [a1,..., an]	Function Application, which evaluates to v if name is bound to function closure cl([var1,...,varn], e, env), [a1,...,an] evaluates to [v1,...,vn], and evaluating e in env extended with the bindings obtained by binding [var1,...,varn] to [v1,...,vn] evaluates to v.

OOP

Expression	**Semantics**
c	Class value, which contains the class name, list of fields, and current environment
o	Object value, which contains the class type, object name, and current environment.

class class_name := {
  [T1 var1,... ,Tn varn]
  func<T1’> func_name1 := e1
  func<T2’> func_name2 := e2
  ...
  func<Tk’> func_namek := ek
}
	Class definition, which evaluates to a new environment env by extending the current environment to include class_name bound to the class value containing the list of variables [var1,...,varn] and the environment containing the functions [func_name1,...,func_namek] bound to their values after evaluating [e1,...,ek] to [v1,...,vk].

class_name var_name := constr [e1, e2, ... , en]	Class object instantiation, which evaluates to a new environment env by extending the current environment with var_name bound to the object of type class_name, with fields varn bound to vn for all n if en evaluates to vn of type Tn in the current environment.
class_name.func_name	Calling a function, which evaluates to cl if func_name evaluates to closure cl in the environment of class_name.
var_name.field_name	Accessing a field, which evaluates to v if in the object to which var_name is bound to, field_name is bound to v.
class_name var_name := constr [e1, e2, ... , en] in e	Class object instantiation expression, which evaluates to v if constr [e1,...,en] evaluates to an object of type class_name v1, e evaluates to v in the environment extended from the current environment to include the binding v1 to var_name.

Other expressions:

Expression	Semantics
u	Unit literal
print [a1]	Evaluates to (). Prints the string representation of v if a1 evaluates to v.
print_endl [a1]	Evaluates to (). Prints the string representation of v concatenated with the newline character if a1 evaluates to v.