Pure Garbage: The Compost Language Reference Manual

October 18, 2023

```
val name_email_map : (string * string) list =
[("Roger Burtonpatel", "roger.burtonpatel@tufts.edu");
 ("Randy Dang", "randy.dang@tufts.edu");
 ("Jasper Geer", "jasper.geer@tufts.edu");
 ("Jackson Warhover", "jackson.warhover@tufts.edu")]
```

rule	::=	(*nonterminal* `terminal`)
	|	{ *other-rule* }
literal	::=	*integer-literal*
	|	*symbol-literal*
	|	*boolean-literal*
	|	*unit-literal*
integer-literal	::=	token composed only of digits, possibly prefixed with a `+` or `-`.
symbol-literal	::=	`'`{ *symbol-character* }`'` *symbol-character* ::= any unicode code point other than `'` and the backslash character unless escaped with a backslash.
boolean-literal	::=	`true` | `false` *unit-literal* ::= `unit`
name	::=	any token that is not an *int-lit*, does not contain whitespace, a `'`, bracket, or parenthesis, and is not a reserved word.
type-expression	::=	*function-type*
	|	*int-type*
	|	*bool-type*
	|	*sym-type*
	|	*unit-type*
	|	*datatype*

int-type ::= `int` *bool-type* ::= `bool` *sym-type* ::= `sym` *unit-type* ::= `unit`

function-type ::= (`->` ({ *type* }) *type*)

datatype ::= *name*

datatype-definition ::= (`datatype` *name* ({ *variant-constructor-definition* })) *variant-constructor-definition*
::= (*name* ({ *type-expression* }))

expr ::= *literal*
| *case-expression*
| *if-expression*
| *begin-expression*
| *apply-expression*
| *let-expression*
| *dup-expression*
| *name-expression*

case-expression ::= (`case` *expr* ({ *case-branch* })) *case-branch* ::= (*pattern* *expr*) *pattern*
::= (*name* { *name* | _ })
| _

if-expression ::= (`if` *expr* *expr* *expr*)

begin-expression ::= (`begin` { *expr* })

apply-expression ::= (*expr* { *expr* })

let-expression ::= (`let` ({ *let-binding* }) *expr*) *let-binding* ::= (*name* *expr*)

name-expression ::= *name*

The type of this expression is $\Gamma[\texttt{n}]$.

dup-expression ::= (`dup` *name*)

The type of this expression is $\Gamma[\texttt{n}]$.

def ::= *val-binding*
| *function-definition*
| *datatype-definition*
| *type-annotation*
| *use-declaration*

type-annotation	::= (`:` *name type-expression*)
val-binding	::= (`val` *name exp*)
function-definition	::= (`define` *name* ({ *name* }) *exp*)
use-declaration	::= (`use` *filename*)
program	::= { *def* } *end-of-file*

(: ¡ (-¿ (int int) bool)) (: ¡= (-¿ (int int) bool))