

# (g)ROOT

## Language Reference Manual

Samuel Russo   Amy Bui   Eliza Encherian  
Zachary Goldstein   Nickolas Gravel

February 27, 2022

## Contents

<b>1</b>	<b>Intro</b>	<b>3</b>
1.1	How to read manual . . . . .	3
<b>2</b>	<b>Lexical Convention</b>	<b>4</b>
2.1	Blanks . . . . .	4
2.2	Comments . . . . .	4
2.3	Identifiers . . . . .	4
2.4	Integer Literals . . . . .	4
2.5	Boolean Literals . . . . .	4
2.6	Character Literals . . . . .	5
2.7	Operators . . . . .	5
2.8	Keywords . . . . .	5
<b>3</b>	<b>Values</b>	<b>6</b>
3.1	Base Values . . . . .	6
3.2	Functions . . . . .	6
3.3	Leaf . . . . .	6
3.4	Tree . . . . .	6
<b>4</b>	<b>Names</b>	<b>7</b>
4.1	Base Values . . . . .	7
4.2	Functions . . . . .	7
<b>5</b>	<b>Constants</b>	<b>7</b>
<b>6</b>	<b>Expressions</b>	<b>7</b>
6.1	. . . . .	7
6.2	Lambda Expression . . . . .	7



# 1 Intro

[Ocaml LRM](#)

## 1.1 How to read manual

The syntax of the language will be given in BNF-like notation. Non-terminal symbol will be in italic font *like-this*, square brackets [ ... ] denote optional components, curly braces { ... } denote zero or more repetitions of the enclosed component, and parenthesis ( ... ) denote a grouping.

## 2 Lexical Convention

### 2.1 Blanks

The following characters are considered as **blanks**: space, horizontal tab (`\t`), newline character (`\n`), and carriage return (`\r`).

Blanks separate adjacent identifiers, literals, expressions, and keywords. They are otherwise ignored.

### 2.2 Comments

Comments are introduced with two adjacent characters (`;` and terminated by two adjacent characters `;`). Nested comments are currently not allowed.

```
(; This is a comment. ;)
```

### 2.3 Identifiers

Identifiers are sequences of letters, digits, and underscore characters (`_`), starting with a letter. Letters consist of the 26 lowercase and 26 uppercase characters from the ASCII set.

$$\langle \textit{ident} \rangle ::= \textit{letter} \ ( \ \textit{letter} \ | \ \textit{digit} \ | \ \_ \ )$$
$$\langle \textit{letter} \rangle ::= \text{a} \dots \text{z} \ | \ \text{A} \dots \text{Z}$$
$$\langle \textit{digit} \rangle ::= 0 \dots 9$$

### 2.4 Integer Literals

An integer literal is a decimal, represented by a sequence of one or more digits, optionally preceded by a minus sign.

$$\langle \textit{integer-literal} \rangle ::= [-] \ \textit{digit} \ \{ \ \textit{digit} \ \}$$
$$\langle \textit{digit} \rangle ::= 0 \dots 9$$

### 2.5 Boolean Literals

Boolean literals are represented by two adjacent characters; the first is the octothorp character (`#`), and it is immediately followed by either the `t` or the `f` character.

$$\langle \textit{boolean-literal} \rangle ::= \# \ ( \ \text{t} \ | \ \text{f} \ )$$

## 2.6 Character Literals

Character literals are a single character enclosed by two ' (single-quote) characters.

## 2.7 Operators

All of the following operators are prefix characters or prefixed characters read as single token. Binary operators are expected to be followed by two expressions, unary operators are expected to be followed by one expression.

$\langle operator \rangle ::= ( unary - operator \mid binary - operator )$

$\langle unary-operator \rangle ::= !$

$\langle binary-operator \rangle ::= + \mid - \mid * \mid / \mid \text{mod}$   
 $\mid == \mid < \mid > \mid \leq \mid \geq \mid !=$   
 $\mid \&\& \mid ||$

## 2.8 Keywords

The below identifiers are reserved keywords and cannot be used otherwise:

if	val	let
leaf?	elm	tree
cld	sib	lambda

The following character sequence are also keywords:

==	+	&&	>	'
!=	-		mod	#t
<=	*	!	(	#f
>=	/	<	)	

## 3 Values

### 3.1 Base Values

#### 3.1.1 Integer numbers

Integer values are integer numbers in range from  $-2^{32}$  to  $2^{32} - 1$ , similar to LLVM's integers, and may support a wider range of integer values on other machines, such as  $-2^{64}$  to  $2^{64} - 1$  on a 64-bit machine.

#### 3.1.2 Boolean values

Booleans have two values. `#t` evaluates to the boolean value `true`, and `#f` evaluates to the boolean value `false`.

#### 3.1.3 Characters

Character values are 8-bit integers between 0 and 255, and follow ASCII standard.

### 3.2 Functions

Functional values are mappings from values to value.

### 3.3 Leaf

### 3.4 Tree

## 4 Names

### 4.1 Base Values

### 4.2 Functions

## 5 Constants

## 6 Expressions

$\langle expr \rangle$  ::= *literal*  
| *ident*  
| *unary-operator* *expr*  
| ( *binary-operator* *expr* *expr* )  
| ( *ident* *expr-list* )  
| ( **val** *ident* *expr* )  
| ( **let** *ident* *expr* *expr* )  
| ( **if** *expr* *expr* *expr* )  
| ( **lambda** ( {*argument*} ) *expr* )

$\langle literal \rangle$  ::= *integer-literal* | *boolean-literal* | *character* | **leaf**

$\langle argument \rangle$  ::=  $\epsilon$   
| *ident* :: *argument*

### 6.1

### 6.2 Lambda Expression

## 7 Functions