

Languages-beta: OC-L-02-Values *

The P_{LAN}CompS Project

OC-L-02-Values.cbs | PLAIN | PRETTY

OUTLINE

2 Values

- Base values
 - Integer numbers
 - Floating-point numbers
 - Characters
 - Character strings
- Tuples
- Records
- Arrays
- Variant values
- Functions

Language "OCaml Light"

2 Values

The comments below are excerpts from section 7.2 of The OCaml System, release 4.06.

Type implemented-values
↔ null-type

- | booleans
- | implemented-integers
- | implemented-floats
- | implemented-characters
- | implemented-strings
- | implemented-tuples
- | implemented-lists
- | implemented-records
- | implemented-references
- | implemented-vectors
- | implemented-variants
- | implemented-functions

*Suggestions for improvement: plancomps@gmail.com.
Reports of issues: <https://github.com/plancomps/CBS-beta/issues>.

Base values

Integer numbers

Integer values are integer numbers from $-2^{\{30\}}$ to $2^{\{30\}}-1$, that is -1073741824 to 1073741823. The implementation may support a wider range of integer values (...).

Type **implemented-integers**
 \rightsquigarrow **integers**

Funcon **implemented-integer**(*I* : **integers**) : \Rightarrow **implemented-integers**
 \rightsquigarrow *I*

Assert **is-equal**(
 null,
 implemented-integer(*N* : **bounded-integers**(-1073741824, 1073741823)))
 == false

Funcon **implemented-integers-width** : \Rightarrow **natural-numbers**
 \rightsquigarrow 31

Funcon **implemented-integer-literal**(*IL* : **strings**) : \Rightarrow **implemented-integers**
 \rightsquigarrow **implemented-integer decimal-natural**(*IL*)

Funcon **implemented-bit-vector**(*I* : **implemented-integers**)
 : \Rightarrow **bit-vectors**(**implemented-integers-width**)
 \rightsquigarrow **integer-to-bit-vector**(*I*, **implemented-integers-width**)

Floating-point numbers

Floating-point values are numbers in floating-point representation. The current implementation uses double-precision floating-point numbers conforming to the IEEE 754 standard, with 53 bits of mantissa and an exponent ranging from -1022 to 1023.

Type **implemented-floats**

Funcon **implemented-floats-format** : \Rightarrow **float-formats**
 \rightsquigarrow **binary64**

Funcon **implemented-float-literal**(*FL* : **strings**) : \Rightarrow **implemented-floats**

Characters

Character values are represented as 8-bit integers between 0 and 255. Character codes between 0 and 127 are interpreted following the ASCII standard. The current implementation interprets character codes between 128 and 255 following the ISO 8859-1 standard.

Type **implemented-characters** <: **characters**

Type **implemented-character-points**
 \rightsquigarrow **bounded-integers**(0, 255)

Funcon **implemented-character**(*C* : **characters**) : \Rightarrow **implemented-characters?**
 \rightsquigarrow **ascii-character** [*C*]

Character strings

String values are finite sequences of characters. The current implementation supports strings containing up to $2^{24} - 5$ characters (16777211 characters); (...)

Type `implemented-strings <: lists(implemented-characters)`

Funcon `implemented-string(L : lists(implemented-characters)) : \Rightarrow implemented-strings?`
 `\rightsquigarrow when-true(is-less-or-equal(length list-elements L, 16777211), L)`

Tuples

Tuples of values are written (*v*₁, ..., *v*_{*n*}), standing for the *n*-tuple of values *v*₁ to *v*_{*n*}. The current implementation supports tuples of up to $2^{22} - 1$ elements (4194303 elements).

Type `implemented-tuples <: tuples(implemented-values*)`
 `\rightsquigarrow tuples(values*)`

Funcon `implemented-tuple(T : tuples(values*)) : \Rightarrow implemented-tuples?`
 `\rightsquigarrow when-true(is-less-or-equal(length tuple-elements T, 4194303), T)`

In OCaml Light, the unit value is represented by `tuple()`.

In OCaml Light, lists are written [*v*₁; ... ; *v*_{*n*}], and their values are represented by list values in CBS.

Type `implemented-lists <: lists(implemented-values)`
 `\rightsquigarrow lists(values)`

Funcon `implemented-list(L : lists(values)) : \Rightarrow implemented-lists?`
 `\rightsquigarrow when-true(is-less-or-equal(length list-elements L, 4194303), L)`

Records

Record values are labeled tuples of values. The record value written { field₁ = *v*₁; ... ; field_{*n*} = *v*_{*n*} } associates the value *v*_{*i*} to the record field field_{*i*}, for *i* = 1 ... *n*. The current implementation supports records with up to $2^{22} - 1$ fields (4194303 fields).

Type `implemented-records <: records(implemented-values)`
 `\rightsquigarrow records(values)`

Funcon `implemented-record(R : records(implemented-values)) : \Rightarrow implemented-records?`
 `\rightsquigarrow when-true(is-less-or-equal(length map-elements record-map R, 4194303), R)`

In OCaml Light, records are non-mutable, and references are represented by mutable variables.

Type `implemented-references \rightsquigarrow variables`

Arrays

Arrays are finite, variable-sized sequences of values of the same type. The current implementation supports arrays containing up to $2^{22} - 1$ elements (4194303 elements) unless the elements are floating-point numbers (2097151 elements in this case); (...)

Type `implemented-vectors` <: `vectors(implemented-values)`
 \rightsquigarrow `vectors(values)`

Funcon `implemented-vector`(V : `vectors(implemented-values)`) : \Rightarrow `implemented-vectors`?
 \rightsquigarrow `when-true(is-less-or-equal(length vector-elements V , 4194303), V)`

Variant values

Variant values are either a constant constructor, or a pair of a non-constant constructor and a value. The former case is written `constr`; the latter case is written `(v1, ..., vn)`, where the v_i are said to be the arguments of the non-constant constructor `constr`. The parentheses may be omitted if there is only one argument. (...) The current implementation limits each variant type to have at most 246 non-constant constructors and $2^{30}-1$ constant constructors.

Type `implemented-variants` <: `variants(implemented-values)`
 \rightsquigarrow `variants(values)`

Funcon `implemented-variant`(V : `variants(implemented-values)`) : \Rightarrow `implemented-variants`
 \rightsquigarrow V

Functions

Functional values are mappings from values to values.

Type `implemented-functions` <: `functions(implemented-values, implemented-values)`
 \rightsquigarrow `functions(values, values)`

Funcon `implemented-function`(F : `functions(implemented-values, implemented-values)`)
 : \Rightarrow `implemented-functions`
 \rightsquigarrow F