Languages-beta: OC-L-02-Values

The PLanCompS Project

`Languages-beta/OC-L/OC-L-02-Values/OC-L-02-Values.cbs`*

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-c

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 ⇝ integers

Funcon implemented-integer(*I* : integers) : ⇒ implemented-integers
⇝ *I*

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

Funcon implemented-integers-width : ⇒ natural-numbers
⇝ 31

*Suggestions for improvement: `plancomps@gmail.com`.
Issues: `https://github.com/plancomps/CBS-beta/issues`.

Funcon implemented-integer-literal(*IL* : strings) : ⇒ implemented-integers
⇝ implemented-integer decimal-natural(*IL*)

Funcon implemented-bit-vector(*I* : implemented-integers) : ⇒ bit-vectors(implemented-integers-width)
⇝ 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 : ⇒ float-formats
⇝ binary64

Funcon implemented-float-literal(*FL* : strings) : ⇒ 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 ⇝ bounded-integers(0, 255)

Funcon implemented-character(*C* : characters) : ⇒ implemented-characters ?
⇝ 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)) : ⇒ implemented-strings ?
⇝ 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 *) ⇝ tuples(values *)

Funcon implemented-tuple(*T* : tuples(values *)) : ⇒ implemented-tuples ?
⇝ 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) ⇝ lists(values)

Funcon implemented-list(*L* : lists(values)) : ⇒ implemented-lists ?
⇝ when-true(is-less-or-equal(length list-elements *L*,
4194303),
L)

Records

Record values are labeled tuples of values. The record value written { field_1 = 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) ⇝ records(values)

Funcon implemented-record(*R* : records(implemented-values)) : ⇒ implemented-records ?
⇝ 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 ⇝ 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) ⇝ vectors(values)

Funcon implemented-vector(*V* : vectors(implemented-values)) : ⇒ implemented-vectors ?
⇝ 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 vi 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) ⇝ variants(values)

Funcon implemented-variant(V : variants(implemented-values)) : ⇒ implemented-variants ⇝ V

Functions

Functional values are mappings from values to values.

Type implemented-functions <: functions(implemented-values, implemented-values) ⇝ functions(values, values)

Funcon implemented-function(F : functions(implemented-values, implemented-values)) : ⇒ implemented-func ⇝ F