Languages-beta: OC-L-01-Lexical-Conventions *

The PLanCompS Project

`OC-L-01-Lexical-Conventions.cbs` | PLAIN | PRETTY

OUTLINE

Language "OCaml Light"

1 Lexical conventions

Identifiers

Lexis *I* : ident ::= capitalized-ident | lowercase-ident

CI : capitalized-ident ::= uppercase (uppercase | lowercase | decimal | '_' | ''')*

LI : lowercase-ident ::= lowercase (uppercase | lowercase | decimal | '_' | ''')*
| '_' (uppercase | lowercase | decimal | '_' | ''')+

uppercase ::= 'A'–'Z'

lowercase ::= 'a'–'z'

decimal ::= '0'–'9'

Semantics id⟦ _ : ident ⟧ : ids

Rule id⟦ *I* ⟧ = "*I*"

---

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

Integer literals

Syntax *IL* : integer-literal ::= '-'?_natural-literal

NL : natural-literal ::= decimal-plus
| ('0x' | '0X') hexadecimal-plus
| ('0o' | '0O') octal-plus
| ('0b' | '0B') binary-plus

Lexis *DP* : decimal-plus ::= decimal$^+$

HP : hexadecimal-plus ::= (decimal | 'A'–'F' | 'a'–'f')$^+$

OP : octal-plus ::= ('0'–'7')$^+$

BP : binary-plus ::= ('0' | '1')$^+$

Semantics integer-value⟦ _ : integer-literal ⟧ : ⇒ implemented-integers

Rule integer-value⟦ '-' *NL* ⟧ = integer-negate(integer-value⟦ *NL* ⟧)

Rule integer-value⟦ *DP* ⟧ = implemented-integer decimal-natural("*DP*")

Floating-point literals

Syntax *FL* : float-literal ::= '-'?_non-negative-float-literal

NNFL : non-negative-float-literal ::= decimal-plus_'.'_decimal-plus
| decimal-plus_'.'
| decimal-plus_'.'_decimal-plus_float-exponent
| decimal-plus_'.'_float-exponent
| decimal-plus_float-exponent

FE : float-exponent ::= ('e' | 'E')_('+' | '-')?_decimal-plus

Rule ⟦ DP_1 '.' DP_2 ⟧ : non-negative-float-literal = ⟦ DP_1 '.' DP_2 'e' '1' ⟧

Rule ⟦ *DP* '.' ⟧ : non-negative-float-literal = ⟦ *DP* '.' '0' 'e' '1' ⟧

Rule ⟦ *DP* '.' *FE* ⟧ : non-negative-float-literal = ⟦ *DP* '.' '0' *FE* ⟧

Rule ⟦ *DP* *FE* ⟧ : non-negative-float-literal = ⟦ *DP* '.' '0' *FE* ⟧

Rule ⟦ 'e' '+' *DP* ⟧ : float-exponent = ⟦ 'e' *DP* ⟧

Rule ⟦ 'E' '+' *DP* ⟧ : float-exponent = ⟦ 'e' *DP* ⟧

Rule ⟦ 'E' '-' *DP* ⟧ : float-exponent = ⟦ 'e' '-' *DP* ⟧

Semantics float-value⟦ _ : float-literal ⟧ : ⇒ implemented-floats

float-value⟦ _ ⟧ is unspecified if the literal value is not representable in floats(implemented-floats-format).

Rule float-value⟦ '-' *NNFL* ⟧ =
float-negate(implemented-floats-format, float-value⟦ *NNFL* ⟧)

Rule float-value⟦ DP_1 '.' DP_2 'e' DP_3 ⟧ =
decimal-float(
implemented-floats-format, "DP_1", "DP_2", "DP_3")

Rule float-value⟦ DP_1 '.' DP_2 'e' '-' DP_3 ⟧ =
decimal-float(
implemented-floats-format, "DP_1", "DP_2", cons('-', "DP_3"))

Character literals

Syntax *CL* : char-literal ::= '' _regular-char_ ''
| '' _escape-sequence_ ''
ES : escape-sequence ::= '\' _escaped-char
| '\' _escaped-char-code
Lexis *RC* : regular-char ::= ~('' | '\')
EC : escaped-char ::= '\' | '"' | '' | 'n' | 't' | 'b' | 'r' | ' '
ECC : escaped-char-code ::= decimal decimal decimal

Semantics character-value⟦ _ : char-literal ⟧ : ⇒ implemented-characters
Rule character-value⟦ '' *RC* '' ⟧ = ascii-character("*RC*")
Rule character-value⟦ '' *ES* '' ⟧ = capture⟦ *ES* ⟧

Semantics capture⟦ _ : escape-sequence ⟧ : implemented-characters
Rule capture⟦ '\' '\' ⟧ = backslash
Rule capture⟦ '\' '' ⟧ = ''
Rule capture⟦ '\' 'n' ⟧ = line-feed
Rule capture⟦ '\' 't' ⟧ = horizontal-tab
Rule capture⟦ '\' 'b' ⟧ = backspace
Rule capture⟦ '\' 'r' ⟧ = carriage-return
Rule capture⟦ '\' *ECC* ⟧ =
checked implemented-character unicode-character decimal-natural("*ECC*")

String literals

Syntax *SL* : string-literal ::= '"' _string-character-star_ '"'
SCS : string-character-star ::= string-character _string-character-star
| ()
SC : string-character ::= regular-string-char
| escape-sequence
Lexis *RSC* : regular-string-char ::= ~('"' | '\')

Semantics string-value⟦ _ : string-literal ⟧ : ⇒ implemented-strings
Rule string-value⟦ '"' *SCS* '"' ⟧ =
checked implemented-string [string-chars⟦ *SCS* ⟧]

Semantics string-chars⟦ _ : string-character-star ⟧ : ⇒ implemented-characters*
Rule string-chars⟦ ⟧ =
Rule string-chars⟦ *SC* *SCS* ⟧ = string-capture⟦ *SC* ⟧, string-chars⟦ *SCS* ⟧

Semantics string-capture⟦ _ : string-character ⟧ : implemented-characters
Rule string-capture⟦ *RSC* ⟧ = ascii-character("*RSC*")
Rule string-capture⟦ *ES* ⟧ = capture⟦ *ES* ⟧

Prefix and infix symbols

Lexis *PS* : prefix-symbol ::= '!' operator-char*
| ('?' | '~') operator-char+

operator-char ::= '!' | '$' | '%' | '&' | '*' | '+' | '-' | '.' | '/'
| ':' | '<' | '=' | '>' | '?' | '@' | '^' | '|' | '~'

operator-char-not-asterisk ::= '!' | '$' | '%' | '&' | '+' | '-' | '.' | '/'
| ':' | '<' | '=' | '>' | '?' | '@' | '^' | '|' | '~'

operator-char-not-bar ::= '!' | '$' | '%' | '&' | '*' | '+' | '-' | '.' | '/'
| ':' | '<' | '=' | '>' | '?' | '@' | '^' | '~'

operator-char-not-ampersand ::= '!' | '$' | '%' | '*' | '+' | '-' | '.' | '/'
| ':' | '<' | '=' | '>' | '?' | '@' | '^' | '|' | '~'

Keywords

Lexis keyword ::= 'and' | 'as' | 'assert' | 'asr' | 'begin' | 'class'
| 'constraint' | 'do' | 'done' | 'downto' | 'else' | 'end'
| 'exception' | 'external' | 'false' | 'for' | 'fun' | 'function'
| 'functor' | 'if' | 'in' | 'include' | 'inherit' | 'initializer'
| 'land' | 'lazy' | 'let' | 'lor' | 'lsl' | 'lsr'
| 'lxor' | 'match' | 'method' | 'mod' | 'module' | 'mutable'
| 'new' | 'nonrec' | 'object' | 'of' | 'open' | 'or'
| 'private' | 'rec' | 'sig' | 'struct' | 'then' | 'to'
| 'true' | 'try' | 'type' | 'val' | 'virtual' | 'when'
| 'while' | 'with'