

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

The P_LanCompS Project

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

OUTLINE

1 Lexical conventions

- Identifiers
- Integer literals
- Floating-point literals
- Character literals
- String literals
- Prefix and infix symbols
- Keywords

Language "OCaml Light"

1 Lexical conventions

Identifiers

Lexis $I : \text{ident} ::= \text{capitalized-ident} \mid \text{lowercase-ident}$

$CI : \text{capitalized-ident} ::= \text{uppercase} (\text{uppercase} \mid \text{lowercase} \mid \text{decimal} \mid \text{'_'} \mid \text{'\''})^*$

$LI : \text{lowercase-ident} ::= \text{lowercase} (\text{uppercase} \mid \text{lowercase} \mid \text{decimal} \mid \text{'_'} \mid \text{'\''})^*$
 $\mid \text{'_'} (\text{uppercase} \mid \text{lowercase} \mid \text{decimal} \mid \text{'_'} \mid \text{'\''})^+$

$\text{uppercase} ::= \text{'A'} - \text{'Z'}$

$\text{lowercase} ::= \text{'a'} - \text{'z'}$

$\text{decimal} ::= \text{'0'} - \text{'9'}$

Semantics $\text{id}[_ : \text{ident}] : \text{ids}$

Rule $\text{id}[I] = \text{"I"}$

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

Integer literals

Syntax $IL : \text{integer-literal} ::= \text{'-'?}_\text{natural-literal}$
 $NL : \text{natural-literal} ::= \text{decimal-plus}$
 $\quad \quad \quad | (\text{'0x' } | \text{'0X'}) \text{ hexadecimal-plus}$
 $\quad \quad \quad | (\text{'0o' } | \text{'0O'}) \text{ octal-plus}$
 $\quad \quad \quad | (\text{'0b' } | \text{'0B'}) \text{ binary-plus}$
Lexis $DP : \text{decimal-plus} ::= \text{decimal}^+$
 $HP : \text{hexadecimal-plus} ::= (\text{decimal} | \text{'A'-'F' } | \text{'a'-'f'})^+$
 $OP : \text{octal-plus} ::= (\text{'0'-'7'})^+$
 $BP : \text{binary-plus} ::= (\text{'0' } | \text{'1'})^+$

Semantics $\text{integer-value}[_ : \text{integer-literal}] : \Rightarrow \text{implemented-integers}$
Rule $\text{integer-value}[\text{'-'} NL] = \text{integer-negate}(\text{integer-value}[NL])$
Rule $\text{integer-value}[DP] = \text{implemented-integer decimal-natural}("DP")$

Floating-point literals

Syntax $FL : \text{float-literal} ::= \text{'-'?}_\text{non-negative-float-literal}$
 $NNFL : \text{non-negative-float-literal} ::= \text{decimal-plus}_\text{'.'}_\text{decimal-plus}$
 $\quad \quad \quad | \text{decimal-plus}_\text{'.'}$
 $\quad \quad \quad | \text{decimal-plus}_\text{'.'}_\text{decimal-plus_float-exponent}$
 $\quad \quad \quad | \text{decimal-plus}_\text{'.'}_\text{float-exponent}$
 $\quad \quad \quad | \text{decimal-plus_float-exponent}$
 $FE : \text{float-exponent} ::= (\text{'e' } | \text{'E'})_\text{'+' | '-'?}_\text{decimal-plus}$

Rule $[DP_1 \text{'.'} DP_2] : \text{non-negative-float-literal} = [DP_1 \text{'.'} DP_2 \text{'e' } '1']$
Rule $[DP \text{'.'}] : \text{non-negative-float-literal} = [DP \text{'.'} '0' \text{'e' } '1']$
Rule $[DP \text{'.'} FE] : \text{non-negative-float-literal} = [DP \text{'.'} '0' FE]$
Rule $[DP FE] : \text{non-negative-float-literal} = [DP \text{'.'} '0' FE]$
Rule $[\text{'e' } \text{'+'} DP] : \text{float-exponent} = [\text{'e' } DP]$
Rule $[\text{'E' } \text{'+'} DP] : \text{float-exponent} = [\text{'e' } DP]$
Rule $[\text{'E' } \text{'-'} DP] : \text{float-exponent} = [\text{'e' } \text{'-'} DP]$

Semantics $\text{float-value}[_ : \text{float-literal}] : \Rightarrow \text{implemented-floats}$

$\text{float-value}[_]$ is unspecified if the literal value is not representable in $\text{floats}(\text{implemented-floats-format})$.

Rule $\text{float-value}[\text{'-'} NNFL] =$
 $\quad \text{float-negate}(\text{implemented-floats-format}, \text{float-value}[NNFL])$
Rule $\text{float-value}[DP_1 \text{'.'} DP_2 \text{'e' } DP_3] =$
 $\quad \text{decimal-float}(\text{implemented-floats-format}, "DP_1", "DP_2", "DP_3")$
Rule $\text{float-value}[DP_1 \text{'.'} DP_2 \text{'e' } \text{'-'} DP_3] =$
 $\quad \text{decimal-float}(\text{implemented-floats-format}, "DP_1", "DP_2", \text{cons}(\text{'-'}, "DP_3"))$

Character literals

Syntax **CL : char-literal** ::= **'** *_regular-char_* **'**
| **'** *_escape-sequence_* **'**

```
ES : escape-sequence ::= '\'_escaped-char
                        | '\'_escaped-char-code
```

Lexis $RC : \text{regular-char} ::= \sim ('' | '\backslash')$

$$EC : \text{escaped-char} ::= \backslash \mid " \mid ' \mid n \mid t \mid b \mid r \mid \text{...}$$

$ECC : \text{escaped-char-code} ::= \text{decimal decimal decimal}$

Semantics $\text{character-value}[_ : \text{char-literal}] : \Rightarrow \text{implemented-characters}$

Rule `character-value` [' ' RC ' '] = `ascii-character` ("RC")

Rule $\text{character-value}[\text{' ' } ES \text{' '}] = \text{capture}[ES]$

Semantics `capture` `[_ : escape-sequence]` : implemented-characters

Rule $\text{capture}[\text{'\'} \text{'\'}] = \text{backslash}$

Rule $\text{capture}[\text{'\ '}] = \text{' '}$

Rule `capture[['\n']] = line-feed`

Rule `capture['\t'] = horizontal-tab`

Rule capture['\ 'b'] = backspace

Rule `capture['\r'] = carriage-return`

Rule $\text{capture}[\text{'\'} ECC] =$

```
checked implemented-character unicode-character decimal-natural("ECC")
```

String literals

Syntax $SL : \text{string-literal} ::= \text{"_string-character-star_"}'$

```
SCS : string-character-star ::= string-character_string-character-star
                                | ()
```

$$SC : \text{string-character} ::= \text{regular-string-char} \mid \text{escape-sequence}$$

Lexis $RSC : \text{regular-string-char} ::= \sim (''' \mid '\backslash')$

Semantics $\text{string-value} \llbracket _ : \text{string-literal} \rrbracket : \Rightarrow \text{implemented-strings}$

Rule `string-value` [' ' SCS ' '] =
checked implemented-string [string-chars [SCS]]

Semantics $\text{string-chars} \llbracket _ : \text{string-character-star} \rrbracket : \Rightarrow \text{implemented-characters}^*$

Rule `string-chars` $\llbracket \quad \rrbracket =$

Rule $\text{string-chars}[\![\ SC\ SCS\]\!] = \text{string-capture}[\![\ SC\]\!], \text{string-chars}[\![\ SCS\]\!]$

Semantics `string-capture` $\llbracket _ : \text{string-character} \rrbracket : \text{implemented-characters}$

Rule `string-capture` $\llbracket RSC \rrbracket = \text{ascii-character}("RSC")$

Rule $\text{string-capture} \llbracket ES \rrbracket = \text{capture} \llbracket ES \rrbracket$

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'
```