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

The PLanCompS Project

`Languages-beta/OC-L/OC-L-01-Lexical-Conventions/OC-L-01-Lexical-Conventions.cb`

Language “`OCaml Light`”

1 Lexical conventions

Identifiers

$$
\begin{array}{rcl}
\textit{Lexis}\ I : \mathsf{ident} & ::= & \mathsf{capitalized\text{-}ident} \\
 & | & \mathsf{lowercase\text{-}ident} \\
CI : \mathsf{capitalized\text{-}ident} & ::= & \mathsf{uppercase}\ (\mathsf{uppercase} \mid \mathsf{lowercase} \mid \mathsf{decimal} \mid \texttt{\_} \mid \texttt{'})^{*} \\
LI : \mathsf{lowercase\text{-}ident} & ::= & \mathsf{lowercase}\ (\mathsf{uppercase} \mid \mathsf{lowercase} \mid \mathsf{decimal} \mid \texttt{\_} \mid \texttt{'})^{*} \\
 & | & \texttt{\_}\ (\mathsf{uppercase} \mid \mathsf{lowercase} \mid \mathsf{decimal} \mid \texttt{\_} \mid \texttt{'})^{+} \\
\mathsf{uppercase} & ::= & \texttt{A} - \texttt{Z} \\
\mathsf{lowercase} & ::= & \texttt{a} - \texttt{z} \\
\mathsf{decimal} & ::= & \texttt{0} - \texttt{9}
\end{array}
$$

$$
\begin{array}{l}
\textit{Semantics}\ \mathsf{id}[\![\ \_ : \mathsf{ident}\ ]\!] : \mathsf{ids} \\
\quad \textit{Rule}\ \mathsf{id}[\![\ I\ ]\!] = \\
\qquad \text{“}I\text{”}
\end{array}
$$

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

Integer literals

$$\begin{array}{rcl}
\textit{Syntax}\ \mathit{IL} : \mathsf{integer\text{-}literal} & ::= & \texttt{-}^{?}\_\mathsf{natural\text{-}literal} \\
\mathit{NL} : \mathsf{natural\text{-}literal} & ::= & \mathsf{decimal\text{-}plus} \\
 & \mid & (\texttt{0x} \mid \texttt{0X})\ \mathsf{hexadecimal\text{-}plus} \\
 & \mid & (\texttt{0o} \mid \texttt{0O})\ \mathsf{octal\text{-}plus} \\
 & \mid & (\texttt{0b} \mid \texttt{0B})\ \mathsf{binary\text{-}plus} \\
\textit{Lexis}\ \mathit{DP} : \mathsf{decimal\text{-}plus} & ::= & \mathsf{decimal}^{+} \\
\mathit{HP} : \mathsf{hexadecimal\text{-}plus} & ::= & (\mathsf{decimal} \mid \texttt{A} - \texttt{F} \mid \texttt{a} - \texttt{f})^{+} \\
\mathit{OP} : \mathsf{octal\text{-}plus} & ::= & (\texttt{0} - \texttt{7})^{+} \\
\mathit{BP} : \mathsf{binary\text{-}plus} & ::= & (\texttt{0} \mid \texttt{1})^{+}
\end{array}$$

$$\begin{array}{l}
\textit{Semantics}\ \mathsf{integer\text{-}value}[\![\ \_ : \mathsf{integer\text{-}literal}\ ]\!] : \Rightarrow \mathsf{implemented\text{-}integers} \\
\quad \textit{Rule}\ \mathsf{integer\text{-}value}[\![\ \texttt{-}\ \mathit{NL}\ ]\!] = \\
\qquad \mathsf{integer\text{-}negate}(\mathsf{integer\text{-}value}[\![\ \mathit{NL}\ ]\!]) \\
\quad \textit{Rule}\ \mathsf{integer\text{-}value}[\![\ \mathit{DP}\ ]\!] = \\
\qquad \mathsf{implemented\text{-}integer}\ \ \mathsf{decimal\text{-}natural}(\text{“}\mathit{DP}\text{”})
\end{array}$$

Floating-point literals

$$\begin{array}{rcl}
\textit{Syntax}\ \mathit{FL} : \mathsf{float\text{-}literal} & ::= & \texttt{-}^{?}\_\mathsf{non\text{-}negative\text{-}float\text{-}literal} \\
\mathit{NNFL} : \mathsf{non\text{-}negative\text{-}float\text{-}literal} & ::= & \mathsf{decimal\text{-}plus}\_\texttt{.}\_\mathsf{decimal\text{-}plus} \\
 & \mid & \mathsf{decimal\text{-}plus}\_\texttt{.} \\
 & \mid & \mathsf{decimal\text{-}plus}\_\texttt{.}\_\mathsf{decimal\text{-}plus}\_\mathsf{float\text{-}exponent} \\
 & \mid & \mathsf{decimal\text{-}plus}\_\texttt{.}\_\mathsf{float\text{-}exponent} \\
 & \mid & \mathsf{decimal\text{-}plus}\_\mathsf{float\text{-}exponent} \\
\mathit{FE} : \mathsf{float\text{-}exponent} & ::= & (\texttt{e} \mid \texttt{E})\_(\texttt{+} \mid \texttt{-})^{?}\_\mathsf{decimal\text{-}plus}
\end{array}$$

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$\llbracket$ _ : string-character $\rrbracket$: implemented-characters

 Rule string-capture$\llbracket$ *RSC* $\rrbracket$ =

  ascii-character(“*RSC*”)

 Rule string-capture$\llbracket$ *ES* $\rrbracket$ =

  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  ::=  !
                            |  $
                            |  %
                            |  &
                            |  *
                            |  +
```

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
```