

Languages-beta: OC-L-07-Expressions *

The P_LanCompS Project

OC-L-07-Expressions.cbs | PLAIN | PRETTY

OUTLINE

7 Expressions

Expression sequences and maps

Matching

Value definitions

Language "OCaml Light"

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

7 Expressions

```
Syntax E : expr ::= value-path
      | constant
      | '(' expr ')'
      | 'begin' expr 'end'
      | '(' expr ':' typexpr ')'
      | expr comma-expr+
      | expr '::' expr
      | '[' expr semic-expr* ']'
      | '[' expr semic-expr* ';' ']'
      | '[' |' expr semic-expr* '|]'
      | '[' |' expr semic-expr* ';' '|]'
      | '{ field '=' expr semic-field-expr* '}'
      | '{ field '=' expr semic-field-expr* ';' '}'
      | '{ expr 'with' field '=' expr semic-field-expr* '}'
      | '{ expr 'with' field '=' expr semic-field-expr* ';' '}'
      | expr argument+
      | prefix-symbol expr
      | '-' expr
      | '-.' expr
      | expr infix-op-1 expr
      | expr infix-op-2 expr
      | expr infix-op-3 expr
      | expr infix-op-4 expr
      | expr infix-op-5 expr
      | expr infix-op-6 expr
      | expr infix-op-7 expr
      | expr infix-op-8 expr
      | expr '.' field
      | expr '.'( expr ')'
      | expr '.'( expr ')' '<-' expr
      | 'if' expr 'then' expr ('else' expr)?
      | 'while' expr 'do' expr 'done'
      | 'for' value-name '=' expr ('to' | 'downto') expr 'do' expr 'done'
      | expr ';' expr
      | 'match' expr 'with' pattern-matching
      | 'function' pattern-matching
      | 'fun' pattern+ '->' expr
      | 'try' expr 'with' pattern-matching
      | let-definition 'in' expr
      | 'assert' expr

A : argument ::= expr

PM : pattern-matching ::= pattern '->' expr pattern-expr*
      | '|' pattern '->' expr pattern-expr*

LD : let-definition ::= 'let' ('rec')? let-binding and-let-binding*

LB : let-binding ::= pattern '=' expr
      | value-name pattern+ '=' expr
      | value-name ':' poly-typexpr '=' expr

ALB : and-let-binding ::= 'and' let-binding2

CE : comma-expr ::= ',' expr

SE : semic-expr ::= ';' expr
```

Rule $[(' E ')]: \text{expr} = [E]$
 Rule $[\text{'begin' } E \text{'end' }]: \text{expr} = [E]$
 Rule $[(' E ' : ' T ')]: \text{expr} = [E]$
 Rule $[E_1 E_2 A A^*]: \text{expr} = [((' E_1 E_2 ')) A A^*]$
 Rule $[PS E]: \text{expr} = [((' PS ')) E]$
 Rule $[- E]: \text{expr} = [((' ~ - ')) E]$
 Rule $[- . E]: \text{expr} = [((' ~ - . ')) E]$
 Rule $[E_1 IO-1 E_2]: \text{expr} = [((' IO-1 ')) E_1 E_2]$
 Rule $[E_1 IO-2 E_2]: \text{expr} = [((' IO-2 ')) E_1 E_2]$
 Rule $[E_1 IO-3 E_2]: \text{expr} = [((' IO-3 ')) E_1 E_2]$
 Rule $[E_1 IO-4 E_2]: \text{expr} = [((' IO-4 ')) E_1 E_2]$
 Rule $[E_1 IO-5 E_2]: \text{expr} = [((' IO-5 ')) E_1 E_2]$
 Rule $[E_1 \& E_2]: \text{expr} = [E_1 \&\& E_2]$
 Rule $[E_1 \text{'or' } E_2]: \text{expr} = [E_1 || E_2]$
 Rule $[E_1 IO-8 E_2]: \text{expr} = [((' IO-8 ')) E_1 E_2]$
 Rule $[E_1 . (E_2 ')]: \text{expr} = [\text{'array_get' } E_1 E_2]$
 Rule $[E_1 . (E_2 ') <- E_3]: \text{expr} = [\text{'array_set' } E_1 E_2 E_3]$
 Rule $[\text{'if' } E_1 \text{'then' } E_2]: \text{expr} = [\text{'if' } E_1 \text{'then' } E_2 \text{'else' } ((' '))]$
 Rule $[\text{'fun' } P \text{'->' } E]: \text{expr} = [\text{'function' } P \text{'->' } E]$
 Rule $[\text{'fun' } P P^+ \text{'->' } E]: \text{expr} = [\text{'fun' } P \text{'->' } (\text{'fun' } P^+ \text{'->' } E)]$
 Rule $[[E SE^* ; ']]: \text{expr} = [[E SE^*]]$
 Rule $[[| E SE^* ; ' |]]: \text{expr} = [[| E SE^* |]]$
 Rule $[{ F = E SFE^* ; ' }]: \text{expr} = [{ F = E SFE^* }]$
 Rule $[{ E_1 \text{'with' } F = E_2 SFE^* ; ' }]: \text{expr} =$
 $[{ E_1 \text{'with' } F = E_2 SFE^* }]$
 Rule $[| P \text{'->' } E PE^*]: \text{pattern-matching} = [P \text{'->' } E PE^*]$
 Rule $[VN : PT = E]: \text{let-binding} = [VN = E]$
 Rule $[VN P^+ = E]: \text{let-binding} = [VN = (\text{'fun' } P^+ \text{'->' } E)]$

Semantics `evaluate[_ : expr] : => implemented-values`

Rule `evaluate[VP] = bound(value-name[VP])`

Rule `evaluate[CNST] = value[CNST]`

Rule `evaluate[(' E ':' T ')] = evaluate[E]`

Rule `evaluate[E1 ', E2 CE*] =`
`tuple(evaluate-comma-sequence[E1 ', E2 CE*])`

Rule `evaluate[E1 ':: E2] = cons(evaluate[E1], evaluate[E2])`

Rule `evaluate[[' E SE* ']] = [evaluate-semic-sequence[E SE*]]`

Rule `evaluate[[' [E SE* ']]] =`
`vector(`
`left-to-right-map(`
`allocate-initialised-variable(implemented-values, given),`
`evaluate-semic-sequence[E SE*])`

Rule `evaluate[[' [']]] = vector()`

Rule `evaluate['{ F '=' E SFE* ' }] =`
`record(collateral(evaluate-field-sequence[F '=' E SFE*]))`

Rule `evaluate['{ E1 'with' F '=' E2 SFE* ' }] =`
`record(`
`map-override(`
`evaluate-field-sequence[F '=' E2 SFE*],`
`checked record-map(evaluate[E1]))`

Rule `evaluate[CSTR E] =`
`variant(constr-name[CSTR], evaluate[E])`

Otherwise `evaluate[E1 E2] =`
`apply(evaluate[E1], evaluate[E2])`

Rule `evaluate[E '.' F] =`
`record-select(evaluate[E], field-name[F])`

Rule `evaluate[E1 '&&' E2] =`
`if-true-else(evaluate[E1], evaluate[E2], false)`

Rule `evaluate[E1 '||' E2] =`
`if-true-else(evaluate[E1], true, evaluate[E2])`

Rule `evaluate['if E1 'then' E2 'else' E3] =`
`if-true-else(evaluate[E1], evaluate[E2], evaluate[E3])`

Rule `evaluate['while E1 'do' E2 'done'] =`
`while(evaluate[E1], effect(evaluate[E2]))`

Rule `evaluate['for VN '=' E1 'to' E2 'do' E3 'done'] =`
`effect(`
`left-to-right-map(`
`case-match(pattern-bind(value-name[VN]), evaluate[E3]),`
`integer-sequence(evaluate[E1], evaluate[E2]))`

Rule `evaluate['for VN '=' E1 'downto' E2 'do' E3 'done'] =`
`effect(`
`left-to-right-map(`
`case-match(pattern-bind(value-name[VN]), evaluate[E3]),`
`reverse integer-sequence(evaluate[E2], evaluate[E1]))`

Rule `evaluate[E1 ';' E2] =`
`sequential(effect(evaluate[E1]), evaluate[E2])`

Rule `evaluate['match E 'with' PM] =`
`give(`
`evaluate[E],`
`else(match[PM], throw(qcaml-light-match-failure)))`

Rule `evaluate['function' PM] =`
`function closure(`

Expression sequences and maps

Semantics `evaluate-comma-sequence` [`_` : (`expr comma-expr*`)] : (`⇒ implemented-values`)⁺

Rule `evaluate-comma-sequence` [`E1 ' , ' E2 CE*`] =
`evaluate` [`E1`], `evaluate-comma-sequence` [`E2 CE*`]

Rule `evaluate-comma-sequence` [`E`] = `evaluate` [`E`]

Semantics `evaluate-semicolon-sequence` [`_` : (`expr semic-expr*`)] : (`⇒ implemented-values`)⁺

Rule `evaluate-semicolon-sequence` [`E1 ' ; ' E2 SE*`] =
`evaluate` [`E1`], `evaluate-semicolon-sequence` [`E2 SE*`]

Rule `evaluate-semicolon-sequence` [`E`] = `evaluate` [`E`]

Semantics `evaluate-field-sequence` [`_` : (`field '=' expr semic-field-expr*`)] : (`⇒ envs`)⁺

Rule `evaluate-field-sequence` [`F1 '=' E1 ' ; ' F2 '=' E2 SFE*`] =
{ `field-name` [`F1`] `↦ evaluate` [`E1`] },
`evaluate-field-sequence` [`F2 '=' E2 SFE*`]

Rule `evaluate-field-sequence` [`F '=' E`] = { `field-name` [`F`] `↦ evaluate` [`E`] }

Matching

Semantics `match` [`_` : `pattern-matching`] : (`implemented-values ⇒ implemented-values`)⁺

Rule `match` [`P1 '->' E1 ' | ' P2 '->' E2 PE*`] =
`match` [`P1 '->' E1`], `match` [`P2 '->' E2 PE*`]

Rule `match` [`P '->' E`] = `case-match`(`evaluate-pattern` [`P`], `evaluate` [`E`])

Value definitions

Semantics `define-values` [`_` : `let-definition`] : `⇒ environments`

Rule `define-values` [`'let' LB ALB*`] = `define-values-nonrec` [`LB ALB*`]

Rule `define-values` [`'let rec' LB ALB*`] =
`recursive`(
 `set`(`bound-ids-sequence` [`LB ALB*`]),
 `define-values-nonrec` [`LB ALB*`])

Semantics `define-values-nonrec` [`_` : (`let-binding and-let-binding*`)] : `⇒ environments`

Rule `define-values-nonrec` [`LB1 'and' LB2 ALB*`] =
`collateral`(`define-values-nonrec` [`LB1`], `define-values-nonrec` [`LB2 ALB*`])

Rule `define-values-nonrec` [`P '=' E`] =
`else`(
 `match`(`evaluate` [`E`], `evaluate-pattern` [`P`]),
 `throw`(`ocaml-light-match-failure`))

Semantics `bound-ids-sequence` [`_` : (`let-binding and-let-binding*`)] : `ids`⁺

Rule `bound-ids-sequence` [`LB`] = `bound-id` [`LB`]

Rule `bound-ids-sequence` [`LB1 'and' LB2 ALB*`] =
`bound-id` [`LB1`], `bound-ids-sequence` [`LB2 ALB*`]

Semantics `bound-id` [`_` : `let-binding`] : `ids`

Rule `bound-id` [`VN '=' E`] = `value-name` [`VN`]

Otherwise `bound-id` [`LB`] = `fail`