

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

The PPlanCompS 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 $E : \text{expr} ::=$

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

$A : \text{argument} ::= \text{expr}$

$PM : \text{pattern-matching} ::=$

- $\text{pattern '->' expr pattern-expr}^*$
- $| \text{'|' pattern '->' expr pattern-expr}^*$

$LD : \text{let-definition} ::= \text{'let' } (\text{'rec'})? \text{ let-binding and-let-binding}^*$

$LB : \text{let-binding} ::=$

- $\text{pattern} = \text{expr}$
- $| \text{value-name pattern}^+ = \text{expr}$
- $| \text{value-name} : \text{poly-typexpr} = \text{expr}$

$ALB : \text{and-let-binding} ::= \text{'and' let-binding}$

$CE : \text{comma-expr} ::= , \text{ expr}$

$SE : \text{semic-expr} ::= ; \text{ expr}$

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

Semantics $\text{evaluate}[_ : \text{expr}] : \Rightarrow \text{implemented-values}$

Rule $\text{evaluate}[VP] = \text{bound}(\text{value-name}[VP])$

Rule $\text{evaluate}[CNST] = \text{value}[CNST]$

Rule $\text{evaluate}['(E ':' T ')'] = \text{evaluate}[E]$

Rule $\text{evaluate}[E_1 ', ' E_2 CE^*] =$
 $\text{tuple}(\text{evaluate-comma-sequence}[E_1 ', ' E_2 CE^*])$

Rule $\text{evaluate}[E_1 ':' E_2] = \text{cons}(\text{evaluate}[E_1], \text{evaluate}[E_2])$

Rule $\text{evaluate}['[E SE^* ']'] = [\text{evaluate-semic-sequence}[E SE^*]]$

Rule $\text{evaluate}['[| E SE^* |]'] =$
 $\text{vector}(\text{left-to-right-map}(\text{allocate-initialised-variable}(\text{implemented-values}, \text{given}), \text{evaluate-semic-sequence}[E SE^*]))$

Rule $\text{evaluate}['[| ' |]'] = \text{vector}()$

Rule $\text{evaluate}['\{ F '=' E SFE^* \}'] =$
 $\text{record}(\text{collateral}(\text{evaluate-field-sequence}[F '=' E SFE^*]))$

Rule $\text{evaluate}['\{ E_1 \text{'with'} F '=' E_2 SFE^* \}'] =$
 $\text{record}(\text{map-override}(\text{evaluate-field-sequence}[F '=' E_2 SFE^*], \text{checked record-map}(\text{evaluate}[E_1])))$

Rule $\text{evaluate}[CSTR E] =$
 $\text{variant}(\text{constr-name}[CSTR], \text{evaluate}[E])$

Otherwise $\text{evaluate}[E_1 E_2] =$
 $\text{apply}(\text{evaluate}[E_1], \text{evaluate}[E_2])$

Rule $\text{evaluate}[E '.' F] =$
 $\text{record-select}(\text{evaluate}[E], \text{field-name}[F])$

Rule $\text{evaluate}[E_1 '\&\&' E_2] =$
 $\text{if-true-else}(\text{evaluate}[E_1], \text{evaluate}[E_2], \text{false})$

Rule $\text{evaluate}[E_1 '||' E_2] =$
 $\text{if-true-else}(\text{evaluate}[E_1], \text{true}, \text{evaluate}[E_2])$

Rule $\text{evaluate}['\text{if } E_1 \text{'then'} E_2 \text{'else'} E_3] =$
 $\text{if-true-else}(\text{evaluate}[E_1], \text{evaluate}[E_2], \text{evaluate}[E_3])$

Rule $\text{evaluate}['\text{while } E_1 \text{'do'} E_2 \text{'done'}] =$
 $\text{while}(\text{evaluate}[E_1], \text{effect}(\text{evaluate}[E_2]))$

Rule $\text{evaluate}['\text{for } VN '=' E_1 \text{'to'} E_2 \text{'do'} E_3 \text{'done'}] =$
 $\text{effect}(\text{left-to-right-map}(\text{case-match}(\text{pattern-bind}(\text{value-name}[VN]), \text{evaluate}[E_3]), \text{integer-sequence}(\text{evaluate}[E_1], \text{evaluate}[E_2])))$

Rule $\text{evaluate}['\text{for } VN '=' E_1 \text{'downto'} E_2 \text{'do'} E_3 \text{'done'}] =$
 $\text{effect}(\text{left-to-right-map}(\text{case-match}(\text{pattern-bind}(\text{value-name}[VN]), \text{evaluate}[E_3]), \text{reverse integer-sequence}(\text{evaluate}[E_2], \text{evaluate}[E_1])))$

Rule $\text{evaluate}[E_1 ';' E_2] =$
 $\text{sequential}(\text{effect}(\text{evaluate}[E_1]), \text{evaluate}[E_2])$

Rule $\text{evaluate}['\text{match } E \text{'with'} PM] =$
 $\text{give}(\text{evaluate}[E], \text{else}(\text{match}[PM], \text{throw}(\text{qcamllight-match-failure})))$

Rule $\text{evaluate}['\text{function } PM] =$
 $\text{function closure}(\text{evaluate}[PM], \text{implemented-values}, \text{given})$

Expression sequences and maps

Semantics	$\text{evaluate-comma-sequence}[_ : (\text{expr comma-expr}^*)] : (\Rightarrow \text{implemented-values})^+$
Rule	$\text{evaluate-comma-sequence}[E_1 \text{ ' , ' } E_2 \text{ CE}^*] =$ $\text{evaluate}[E_1], \text{evaluate-comma-sequence}[E_2 \text{ CE}^*]$
Rule	$\text{evaluate-comma-sequence}[E] = \text{evaluate}[E]$
Semantics	$\text{evaluate-semicolon-sequence}[_ : (\text{expr semic-expr}^*)] : (\Rightarrow \text{implemented-values})^+$
Rule	$\text{evaluate-semicolon-sequence}[E_1 \text{ ' ; ' } E_2 \text{ SE}^*] =$ $\text{evaluate}[E_1], \text{evaluate-semicolon-sequence}[E_2 \text{ SE}^*]$
Rule	$\text{evaluate-semicolon-sequence}[E] = \text{evaluate}[E]$
Semantics	$\text{evaluate-field-sequence}[_ : (\text{field '=' expr semic-field-expr}^*)] : (\Rightarrow \text{envs})^+$
Rule	$\text{evaluate-field-sequence}[F_1 \text{ '=' } E_1 \text{ ' ; ' } F_2 \text{ '=' } E_2 \text{ SFE}^*] =$ $\{\text{field-name}[F_1] \mapsto \text{evaluate}[E_1]\},$ $\text{evaluate-field-sequence}[F_2 \text{ '=' } E_2 \text{ SFE}^*]$
Rule	$\text{evaluate-field-sequence}[F \text{ '=' } E] = \{\text{field-name}[F] \mapsto \text{evaluate}[E]\}$

Matching

Semantics	$\text{match}[_ : \text{pattern-matching}] : (\text{implemented-values} \Rightarrow \text{implemented-values})^+$
Rule	$\text{match}[P_1 \text{ '->' } E_1 \text{ ' ' } P_2 \text{ '->' } E_2 \text{ PE}^*] =$ $\text{match}[P_1 \text{ '->' } E_1], \text{match}[P_2 \text{ '->' } E_2 \text{ PE}^*]$
Rule	$\text{match}[P \text{ '->' } E] = \text{case-match}(\text{evaluate-pattern}[P], \text{evaluate}[E])$

Value definitions

Semantics	$\text{define-values}[_ : \text{let-definition}] : \Rightarrow \text{environments}$
Rule	$\text{define-values}[\text{'let' } LB \text{ ALB}^*] = \text{define-values-nonrec}[LB \text{ ALB}^*]$
Rule	$\text{define-values}[\text{'let rec' } LB \text{ ALB}^*] =$ $\text{recursive}(\text{set}(\text{bound-ids-sequence}[LB \text{ ALB}^*]),$ $\text{define-values-nonrec}[LB \text{ ALB}^*])$
Semantics	$\text{define-values-nonrec}[_ : (\text{let-binding and-let-binding}^*)] : \Rightarrow \text{environments}$
Rule	$\text{define-values-nonrec}[LB_1 \text{ 'and' } LB_2 \text{ ALB}^*] =$ $\text{collateral}(\text{define-values-nonrec}[LB_1], \text{define-values-nonrec}[LB_2 \text{ ALB}^*])$
Rule	$\text{define-values-nonrec}[P \text{ '=' } E] =$ $\text{else}(\text{match}(\text{evaluate}[E], \text{evaluate-pattern}[P]),$ $\text{throw}(\text{ocaml-light-match-failure}))$
Semantics	$\text{bound-ids-sequence}[_ : (\text{let-binding and-let-binding}^*)] : \text{ids}^+$
Rule	$\text{bound-ids-sequence}[LB] = \text{bound-id}[LB]$
Rule	$\text{bound-ids-sequence}[LB_1 \text{ 'and' } LB_2 \text{ ALB}^*] =$ $\text{bound-id}[LB_1], \text{bound-ids-sequence}[LB_2 \text{ ALB}^*]$
Semantics	$\text{bound-id}[_ : \text{let-binding}] : \text{ids}$
Rule	$\text{bound-id}[VN \text{ '=' } E] = \text{value-name}[VN]$
Otherwise	$\text{bound-id}[LB] = \text{fail}$