Languages-beta: OC-L-A-Disambiguation

The PLanCompS Project

`Languages-beta/OC-L/OC-L-A-Disambiguation/OC-L-A-Disambiguation.cbs`*

Language "`OCaml Light`"

A Disambiguation

Lexis SDF

// 1 Lexical conventions

// Comments

lexical syntax

LAYOUT = LEX-block-comment
LEX-block-comment = "(" *LEX-comment-part* "*)"

LEX-comment-part = ~[()*]
LEX-comment-part = LEX-asterisk
LEX-comment-part = LEX-left-paren
LEX-comment-part = LEX-right-paren
LEX-comment-part = LEX-block-comment

LEX-asterisk = [*]
LEX-left-paren = [(]
LEX-right-paren = [)]

lexical restrictions

LEX-asterisk -/- [)]
LEX-left-paren -/- [*]

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

context-free restrictions

LAYOUT? -/- [(].[*]

// Identifiers

lexical syntax

```
ident = keyword {reject}
lowercase-ident = keyword {reject}
```

lexical restrictions

```
ident
lowercase-ident
capitalized-ident -/- [A-Za-z0-9_']
```

Syntax SDF

// Integer literals

context-free restrictions

```
integer-literal -/- [0-9eE]
```

Syntax SDF

// Floating-point literals

context-free restrictions

```
float-literal -/- [0-9eE]
```

// String literals

syntax

```
string-character-star ::= string-character_string-character-star {avoid}
```

Lexis SDF

// Keywords

lexical restrictions

"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"
-/- [A-Za-z0-9_]

// Key symbols

infix-op-1 infix-op-2 infix-op-3 infix-op-4
infix-op-5 infix-op-6 infix-op-7 infix-op-8
-/- [!$\%*+-.\/:<=>\?\@\^|\~]
"[" -/- [|]
"|" -/- [|]
":" -/- [:]
";" -/- [\;]

lexical syntax

infix-op-3 = "->" {reject}

infix-op-5 = "<-" {reject}

Syntax SDF

// 4 Type expressions

context-free syntax

typexpr ::= typexpr -> typexpr {right}
typexpr ::= typexpr star-typexpr^{+} {non-assoc}

context-free priorities

typexpr ::= typexpr typeconstr
>
constr-args ::= typexpr star-typexpr*
>
typexpr ::= typexpr star-typexpr^{+}
>
typexpr ::= typexpr -> typexpr

context-free priorities

star-typexpr ::= * typexpr
>

```
typexpr ::= typexpr star-typexpr+
```

// 6 Patterns

context-free syntax

```
pattern ::= pattern | pattern {left}
pattern ::= pattern comma-pattern+ {non-assoc}
pattern ::= pattern :: pattern {right}
```

context-free priorities

```
pattern ::= constr pattern
>
pattern ::= pattern :: pattern
>
pattern ::= pattern comma-pattern+
>
pattern ::= pattern | pattern
>
pattern ::= pattern as value-name
```

context-free priorities

```
{
comma-pattern ::= , pattern
(pattern comma-pattern*)
} >
pattern ::= pattern comma-pattern+
```

// 7 Expressions

context-free syntax

```
expr ::= expr argument+ {non-assoc,avoid}
expr ::= - expr {avoid}
expr ::= expr infix-op-1 expr {right}
expr ::= expr infix-op-2 expr {left}
expr ::= expr infix-op-3 expr {left,prefer}
expr ::= expr :: expr {right}
expr ::= expr infix-op-4 expr {right}
expr ::= expr infix-op-5 expr {left}
expr ::= expr infix-op-6 expr {right}
expr ::= expr infix-op-7 expr {right}
expr ::= expr comma-expr+ {non-assoc}
```

```
expr ::= expr infix-op-8 expr {right}
expr ::= expr . field <- expr{right}
expr ::= expr .( expr ) <- expr{right}
expr ::= expr ; expr {right}
```

context-free priorities

```
argument ::= expr
>
expr ::= prefix-symbol expr
>
expr ::= expr . field
> {
expr ::= expr argument+
expr ::= assert expr
} > {
expr ::= - expr
expr ::= -. expr
} >
expr ::= expr infix-op-1 expr
>
expr ::= expr infix-op-2 expr
>
expr ::= expr infix-op-3 expr
>
expr ::= expr :: expr
>
expr ::= expr infix-op-4 expr
>
expr ::= expr infix-op-5 expr
>
expr ::= expr infix-op-6 expr
>
expr ::= expr infix-op-7 expr
>
expr ::= expr comma-expr+
> {
expr ::= expr . field <- expr
expr ::= expr .( expr ) <- expr
expr ::= expr infix-op-8 expr
} >
expr ::= expr ; expr
```

context-free priorities

```
expr ::= prefix-symbol expr
```

>
expr ::= expr .(expr)
<0> >
expr ::= expr argument+

context-free priorities
{
argument ::= expr
expr ::= expr . field
expr ::= expr .(expr)
expr ::= expr argument+
expr ::= assert expr
expr ::= expr infix-op-1 expr
expr ::= expr infix-op-2 expr
expr ::= expr infix-op-3 expr
expr ::= expr :: expr
expr ::= expr infix-op-4 expr
expr ::= expr infix-op-5 expr
expr ::= expr infix-op-6 expr
expr ::= expr infix-op-7 expr
expr ::= expr comma-expr+
expr ::= expr . field <- expr
expr ::= expr .(expr) <- expr
expr ::= expr infix-op-8 expr
expr ::= expr ; expr
} <0>. > {
expr ::= if expr then expr (else expr)?
expr ::= match expr with pattern-matching
expr ::= function pattern-matching
expr ::= fun pattern+ -> expr
expr ::= try expr with pattern-matching
expr ::= let-definition in expr
}

context-free priorities
{
comma-expr ::= , expr
(expr comma-expr*)
} >
expr ::= expr comma-expr+

context-free priorities
{
expr ::= [expr semic-expr*]
expr ::= [expr semic-expr* ;]
expr ::= [| expr semic-expr* |]

```
expr ::= [| expr semic-expr* ; |]
semic-expr ::= ; expr
(expr semic-expr*)
expr ::= { field = expr semic-field-expr* }
expr ::= { field = expr semic-field-expr* ; }
expr ::= { expr with field = expr semic-field-expr* }
expr ::= { expr with field = expr semic-field-expr* ; }
semic-field-expr ::= ; field = expr
} >
expr ::= expr ; expr
```