Languages-beta: OC-L-06-Patterns *

The PLanCompS Project

`OC-L-06-Patterns.cbs` | PLAIN | PRETTY

OUTLINE

Language "OCaml Light"

6 Patterns

Syntax

P : pattern ::= value-name
| '_'
| constant
| pattern 'as' value-name
| '(' pattern ')'
| '(' pattern ':' typexpr ')'
| pattern '|' pattern
| constr pattern
| pattern comma-pattern^{+}
| '{' field '=' pattern semic-field-pattern* ';'$^{?}$ '}'
| '[' pattern semic-pattern* ';'$^{?}$ ']'
| pattern '::' pattern

CP : comma-pattern ::= ',' pattern

SP : semic-pattern ::= ';' pattern

SFP : semic-field-pattern ::= ';' field '=' pattern

Rule ⟦ '(' *P* ')' ⟧ : pattern = ⟦ *P* ⟧

Rule ⟦ '(' *P* ':' *T* ')' ⟧ : pattern = ⟦ *P* ⟧

Rule ⟦ '{' *F* '=' *P SFP** ';' '}' ⟧ : pattern = ⟦ '{' *F* '=' *P SFP** '}' ⟧

Rule ⟦ '[' *P SP** ';' ']' ⟧ : pattern = ⟦ '[' *P SP** ']' ⟧

---

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

Pattern evaluation

```
Semantics  evaluate-pattern⟦ _ : pattern ⟧ : ⇒ patterns
Rule       evaluate-pattern⟦ VN ⟧ = pattern-bind(value-name⟦ VN ⟧)
Rule       evaluate-pattern⟦ '_' ⟧ = pattern-any
Rule       evaluate-pattern⟦ CNST ⟧ = value⟦ CNST ⟧
Rule       evaluate-pattern⟦ P 'as' VN ⟧ =
             pattern-unite(evaluate-pattern⟦ P ⟧, pattern-bind(value-name⟦ VN ⟧))
Rule       evaluate-pattern⟦ P1 '|' P2 ⟧ =
             pattern-else(evaluate-pattern⟦ P1 ⟧, evaluate-pattern⟦ P2 ⟧)
Rule       evaluate-pattern⟦ CSTR P ⟧ =
             variant(constr-name⟦ CSTR ⟧, evaluate-pattern⟦ P ⟧)
Rule       evaluate-pattern⟦ P1 ',' P2 CP* ⟧ =
             tuple(evaluate-comma-pattern-sequence⟦ P1 ',' P2 CP* ⟧)
Rule       evaluate-pattern⟦ '{' F '=' P SFP* '}' ⟧ =
             pattern closure(
               match-loosely(
                 given,
                 record(map-unite(evaluate-field-pattern-sequence⟦ F '=' P SFP* ⟧))))
Rule       evaluate-pattern⟦ '[' P SP* ']' ⟧ =
             [evaluate-semic-pattern-sequence⟦ P SP* ⟧]
Rule       evaluate-pattern⟦ P1 '::' P2 ⟧ =
             pattern closure(
               if-true-else(
                 is-equal(given, [ ]),
                 fail,
                 collateral(
                   match(head(given), evaluate-pattern⟦ P1 ⟧),
                   match(tail(given), evaluate-pattern⟦ P2 ⟧)))))
```

Pattern sequence evaluation

```
Semantics  evaluate-comma-pattern-sequence⟦ _ : (pattern comma-pattern*) ⟧
             : (⇒ patterns)+
Rule       evaluate-comma-pattern-sequence⟦ P1 ',' P2 CP* ⟧ =
             evaluate-pattern⟦ P1 ⟧, evaluate-comma-pattern-sequence⟦ P2 CP* ⟧
Rule       evaluate-comma-pattern-sequence⟦ P ⟧ = evaluate-pattern⟦ P ⟧

Semantics  evaluate-semic-pattern-sequence⟦ _ : (pattern semic-pattern*) ⟧
             : (⇒ patterns)+
Rule       evaluate-semic-pattern-sequence⟦ P1 ';' P2 SP* ⟧ =
             evaluate-pattern⟦ P1 ⟧, evaluate-semic-pattern-sequence⟦ P2 SP* ⟧
Rule       evaluate-semic-pattern-sequence⟦ P ⟧ = evaluate-pattern⟦ P ⟧
```

Semantics evaluate-field-pattern-sequence⟦ _ : (field '=' pattern semic-field-pattern*) ⟧
: ⇒(maps(ids, patterns))+

Rule evaluate-field-pattern-sequence⟦ F_1 '=' P_1 ';' F_2 '=' P_2 SFP^* ⟧ =
(
{field-name⟦ F_1 ⟧ ↦ evaluate-pattern⟦ P_1 ⟧},
evaluate-field-pattern-sequence⟦ F_2 '=' P_2 SFP^* ⟧)

Rule evaluate-field-pattern-sequence⟦ F '=' P ⟧ =
{field-name⟦ F ⟧ ↦ evaluate-pattern⟦ P ⟧}