Languages-beta:
OC-L-08-Type-and-Exception-Definitions

The PLanCompS Project

`Languages-beta/OC-L/OC-L-08-Type-and-Exception-Definitions/OC-L-08-Type-and-Ex`

Language "`OCaml Light`"

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

8 Type and exception definitions

Syntax *TDS* : type-definition ::= `type` typedef and-typedef*

ATD : and-typedef ::= `and` typedef

TD : typedef ::= type-params? typeconstr-name type-information

TI : type-information ::= type-equation? type-representation? type-constraint*

TE : type-equation ::= `=` typexpr

TR : type-representation ::= `=` `|`? constr-decl bar-constr-decl*
| `=` record-decl

BCD : bar-constr-decl ::= `|` constr-decl

TPS : type-params ::= type-param
| `(` type-param (`,` type-param)* `)`

TP : type-param ::= variance? `'` ident

variance ::= `+`
| `-`

RD : record-decl ::= `{` field-decl (`;` field-decl)* `;`? `}`

CD : constr-decl ::= (constr-name | `[]` | `(::)`) (`of` constr-args)?

CA : constr-args ::= typexpr star-typexpr*

FD : field-decl ::= field-name `:` poly-typexpr

ED : exception-definition ::= `exception` constr-decl
| `exception` constr-name `=` constr

TC : type-constraint ::= `constraint` `'` ident `=` typexpr

Type definitions

Semantics define-types⟦ _ : type-definition ⟧ : ⇒ environments

Rule define-types⟦ `type` *TD* *ATD** ⟧ =
collateral(define-typedefs⟦ *TD* *ATD** ⟧)

Semantics define-typedefs⟦ _ : (typedef and-typedef*) ⟧ : (⇒ environments)+
Rule define-typedefs⟦ TD_1 `and` TD_2 ATD^* ⟧ =
define-typedefs⟦ TD_2 ⟧,
define-typedefs⟦ TD_2 ATD^* ⟧
Rule define-typedefs⟦ $TPS^?$ TCN = CD BCD^* ⟧ =
define-constrs⟦ CD BCD^* ⟧
Rule define-typedefs⟦ $TPS^?$ TCN = RD ⟧ =
map( )
Rule define-typedefs⟦ $TPS^?$ TCN = T ⟧ =
map( )

Semantics define-constrs⟦ _ : (constr-decl bar-constr-decl*) ⟧ : (⇒ environments)+
Rule define-constrs⟦ CD_1 | CD_2 BCD^* ⟧ =
define-constrs⟦ CD_1 ⟧,
define-constrs⟦ CD_2 BCD^* ⟧
Rule define-constrs⟦ CN ⟧ =
{constr-name⟦ CN ⟧ ↦ variant(constr-name⟦ CN ⟧,
tuple( ))}
Rule define-constrs⟦ CN `of` CA ⟧ =
{constr-name⟦ CN ⟧ ↦ function closure(variant(constr-name⟦ CN ⟧,
given))}

Exception definitions

Semantics define-exception⟦ _ : exception-definition ⟧ : ⇒ environments
Rule define-exception⟦ `exception` CD ⟧ =
define-constrs⟦ CD ⟧
Rule define-exception⟦ `exception` CN = $CSTR$ ⟧ =
map( )