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

The PLanCompS Project

`OC-L-08-Type-and-Exception-Definitions.cbs` | PLAIN | PRETTY

OUTLINE

Language "OCaml Light"

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

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

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( )