LAMBDA

MODULE LAMBDA

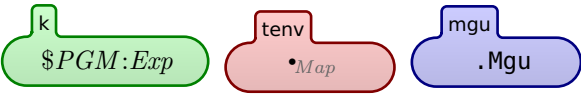SYNTAX $Exp ::= Int$
         | $Bool$
         | Id
         | (Exp) [bracket]
         | $Exp\ Exp$ [strict]
         | $Exp * Exp$ [strict]
         | Exp / Exp [strict]
         | $Exp + Exp$ [strict]
         | $Exp <= Exp$ [strict]
         | lambda Id . Exp [binder]
         | if Exp then Exp else Exp [strict]
         | let Id = Exp in Exp
         | letrec $Id\ Id$ = Exp in Exp
         | mu Id . Exp [binder]

SYNTAX $Type ::=$ int
         | bool
         | $Type$ -> $Type$
         | $(Type)$ [bracket]

SYNTAX $Exp ::= Type$

SYNTAX $KResult ::= Type$

CONFIGURATION:

RULE $\dfrac{I\!:\!Int}{\text{int}}$

RULE $\dfrac{B\!:\!Bool}{\text{bool}}$

RULE

RULE $\dfrac{T1\!:\!Type * T2\!:\!Type}{T1 = \text{int} \curvearrowright T2 = \text{int} \curvearrowright \text{int}}$

RULE $\dfrac{T1\!:\!Type / T2\!:\!Type}{T1 = \text{int} \curvearrowright T2 = \text{int} \curvearrowright \text{int}}$

RULE $\dfrac{T1\!:\!Type + T2\!:\!Type}{T1 = \text{int} \curvearrowright T2 = \text{int} \curvearrowright \text{int}}$

RULE $\dfrac{T1\!:\!Type <= T2\!:\!Type}{T1 = \text{int} \curvearrowright T2 = \text{int} \curvearrowright \text{bool}}$

SYNTAX $Exp ::= Exp$ -> Exp [strict]

RULE

requires fresh $(T\!:\!Type)$

RULE $\dfrac{T1\!:\!Type\ T2\!:\!Type}{T1 = (T2\ \text{->}\ T) \curvearrowright T}$ requires fresh $(T\!:\!Type)$

RULE $\dfrac{\text{if } T\!:\!Type \text{ then } T1\!:\!Type \text{ else } T2\!:\!Type}{T = \text{bool} \curvearrowright T1 = T2 \curvearrowright T1}$

RULE $\dfrac{\text{let } X = E \text{ in } E'}{E'[E\ /\ X]}$ [macro]

RULE $\dfrac{\text{letrec } F\ X = E \text{ in } E'}{\text{let } F = \text{mu } F\ .\ \text{lambda } X\ .\ E \text{ in } E'}$ [macro]

RULE

requires fresh $(T\!:\!Type)$

SYNTAX $K ::= Type = Type$

RULE

RULE

SYNTAX $K ::=$ tenv (Map)

RULE

END MODULE