LAMBDA

MODULE LAMBDA

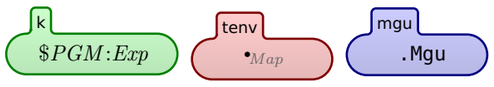
SYNTAX *Exp* ::= *Int*
| *Bool*
| *Id*
| (*Exp*) [bracket]
| *Exp* *Exp* [strict]
| *Exp* * *Exp* [strict]
| *Exp* / *Exp* [strict]
| *Exp* + *Exp* [strict]
| *Exp* <= *Exp* [strict]
| `lambda` *Id* `.` *Exp*
| `if` *Exp* `then` *Exp* `else` *Exp* [strict]
| `let` *Id* = *Exp* `in` *Exp* [strict(2)]
| `letrec` *Id* *Id* = *Exp* `in` *Exp*
| `mu` *Id* `.` *Exp*

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

SYNTAX *Exp* ::= *Type*

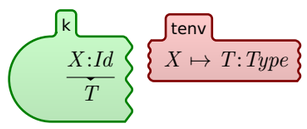
SYNTAX *KResult* ::= *Type*

CONFIGURATION:



RULE $\frac{I:Int}{\texttt{int}}$

RULE $\frac{B:Bool}{\texttt{bool}}$

RULE 

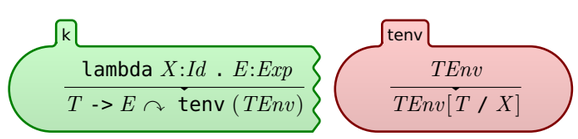
RULE $\frac{T1:Type \texttt{ * } T2:Type}{T1 \texttt{ = int} \curvearrowright T2 \texttt{ = int} \curvearrowright \texttt{int}}$

RULE $\frac{T1:Type \texttt{ / } T2:Type}{T1 \texttt{ = int} \curvearrowright T2 \texttt{ = int} \curvearrowright \texttt{int}}$

RULE $\frac{T1:Type \texttt{ + } T2:Type}{T1 \texttt{ = int} \curvearrowright T2 \texttt{ = int} \curvearrowright \texttt{int}}$

RULE $\frac{T1:Type \texttt{ <= } T2:Type}{T1 \texttt{ = int} \curvearrowright T2 \texttt{ = int} \curvearrowright \texttt{bool}}$

SYNTAX *Exp* ::= *Exp* `->` *Exp* [strict]

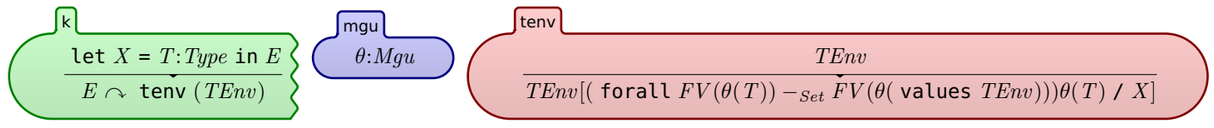
RULE 

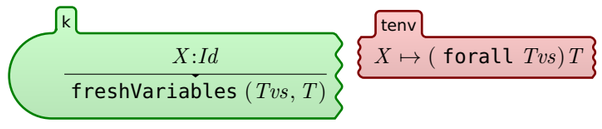
requires `fresh` (*T*:*Type*)

RULE $\frac{T1:Type \quad T2:Type}{T1 \texttt{ = (} T2 \texttt{ -> } T\texttt{)} \curvearrowright T}$ requires `fresh` (*T*:*Type*)

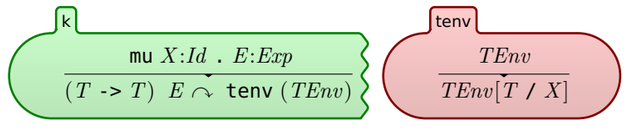
RULE $\frac{\texttt{if } T:Type \texttt{ then } T1:Type \texttt{ else } T2:Type}{T \texttt{ = bool} \curvearrowright T1 \texttt{ = } T2 \curvearrowright T1}$

SYNTAX *TypeSchema* ::= (`forall` *Set*)*Type* [binder]

RULE 

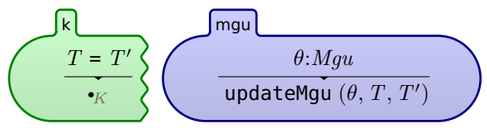
RULE 

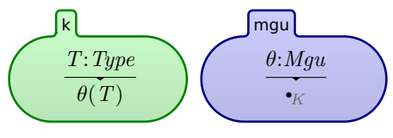
RULE $\frac{\texttt{letrec } F \texttt{ } X \texttt{ = } E \texttt{ in } E'}{\texttt{let } F \texttt{ = mu } F \texttt{ . lambda } X \texttt{ . } E \texttt{ in } E'}$ [macro]

RULE 

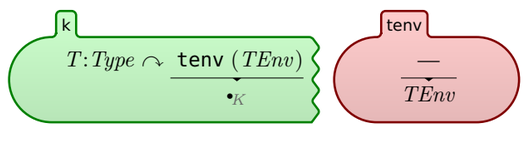
requires `fresh` (*T*:*Type*)

SYNTAX *K* ::= *Type* = *Type*

RULE 

RULE 

SYNTAX *K* ::= `tenv` (*Map*)

RULE 

END MODULE