LAMBDA

MODULE LAMBDA

SYNTAX *Val* ::= *Id*
| λ*Id*.*Exp* [binder(binder())]

SYNTAX *Exp* ::= *Val*
| *Exp* *Exp* [strict(strict())]
| (*Exp*) [bracket(bracket())]

SYNTAX *Variable* ::= *Id*

SYNTAX *KResult* ::= *Val*

RULE $\dfrac{(\lambda X{:}Id.E{:}Exp)\ \ V{:}Val}{E[V \,/\, X]}$

SYNTAX *Val* ::= *Int*
| *Bool*

SYNTAX *Exp* ::= *Exp* * *Exp* [strict(strict())]
| *Exp* / *Exp* [strict(strict())]
| *Exp* + *Exp* [strict(strict())]
| *Exp* <= *Exp* [strict(strict())]

RULE $\dfrac{I1{:}Int * I2{:}Int}{I1 *_{Int} I2}$

RULE $\dfrac{I1{:}Int \,/\, I2{:}Int}{I1 \div_{Int} I2}$

RULE $\dfrac{I1{:}Int + I2{:}Int}{I1 +_{Int} I2}$

RULE $\dfrac{I1{:}Int <= I2{:}Int}{I1 \leq_{Int} I2}$

SYNTAX *Exp* ::= `if` *Exp* `then` *Exp* `else` *Exp* [strict(strict(1))]

RULE $\dfrac{\texttt{if true then}\ E\ \texttt{else}\ \_}{E}$

RULE $\dfrac{\texttt{if false then}\ \_\ \texttt{else}\ E}{E}$

SYNTAX *Exp* ::= `let` *Id* = *Exp* `in` *Exp*

RULE $\dfrac{\texttt{let}\ X\ \texttt{=}\ E\ \texttt{in}\ E'{:}Exp}{(\lambda X.E')\ \ E}$ [macro(macro())]

SYNTAX *Exp* ::= `letrec` *Id* *Id* = *Exp* `in` *Exp*
| μ*Id*.*Exp* [binder(binder())]

RULE $\dfrac{\texttt{letrec}\ F{:}Id\ \ X\ \texttt{=}\ E\ \texttt{in}\ E'}{\texttt{let}\ F\ \texttt{=}\ \mu F.\lambda X.E\ \texttt{in}\ E'}$ [macro(macro())]

RULE $\dfrac{\mu X.E}{E[(\mu X.E) \,/\, X]}$

SYNTAX *Exp* ::= `callcc` *Exp* [strict(strict())]

SYNTAX *Val* ::= `cc` (*K*) [klabel(klabel('cc))]

RULE k $\dfrac{\texttt{callcc}\ V{:}Val \curvearrowright K}{V\ \ \texttt{cc}\ (K)}$

RULE k $\dfrac{\texttt{cc}\ (K)\ \ V \curvearrowright \_}{V \curvearrowright K}$

END MODULE