LAMBDA

MODULE LAMBDA

SYNTAX   *Val* ::= *Id*
       | $\lambda Id.Exp$ [binder(binder())]

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

SYNTAX   *Variable* ::= *Id*

SYNTAX   *KResult* ::= *Val*

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

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 $\dfrac{I1{:}Int \mathbin{/} I2{:}Int}{I1 \div_{Int} I2}$

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 $\dfrac{\textsf{if true then } E \textsf{ else } —}{E}$

RULE $\dfrac{\textsf{if false then } — \textsf{ else } E}{E}$

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

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

SYNTAX   *Exp* ::= letrec *Id* *Id* = *Exp* in *Exp*
       | $\mu Id.Exp$ [binder(binder())]

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

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

END MODULE