

Di seguito trovate alcuni test. La maggior parte devono, per così dire, passare. Alcuni no. Non compaiono esempi per il `Try` che verranno aggiunti presto. Non sono in ordine d'importanza e dopo ciascuno viene indicato il risultato che dovete ottenere sia con la `type_inference` che con la `sem`. La valutazione di `Rec` può avere due valori. Sono riportati entrambi. Attenzione che l'ambiente (qui indicato con `<fun>`) può essere ispezionato e deve contenere le associazioni corrette.

```
Let(Ide "f", Fun(Ide "x",Fun(Ide "y",Val(Ide "x"))),
    Appl(Appl(Val(Ide "f"),Eint 2),Eint 1))
```

```
tipo: etype = TInt
valore: eval = Int 2
```

```
Let(Ide "fact",Rec(Ide "fact", Fun(Ide "x", Ifthenelse(
    Eq(Val(Ide "x"), Eint 0), Eint 1,
    Times(Val(Ide "x"), Appl (Val(Ide "fact"),
                                Diff(Val(Ide "x"), Eint 1)))))),
    Appl(Val(Ide "fact"),Eint 5))
```

```
tipo: etype = TInt
valore: eval = Int 120
```

```
Rec(Ide "fact", Fun(Ide "x", Ifthenelse(
    Eq(Val(Ide "x"), Eint 0), Eint 1,
    Times(Val(Ide "x"), Appl (Val(Ide "fact"),
                                Diff(Val(Ide "x"), Eint 1))))))
```

```
tipo: etype = TFun (TInt, TInt)
valore: eval = Closure
  (Fun (Ide "x",
    Ifthenelse (Eq (Val (Ide "x"), Eint 0), Eint 1,
      Times (Val (Ide "x"),
        Appl (Val (Ide "fact"), Diff (Val (Ide "x"), Eint 1))))),
  <fun>)
valore: eval =
Closure
  (Fun (Ide "x",
    Ifthenelse (Eq (Val (Ide "x"), Eint 0), Eint 1,
      Times (Val (Ide "x"),
        Appl
          (Rec (Ide "fact",
```

```

    Fun (Ide "x",
        Ifthenelse (Eq (Val (Ide "x"), Eint 0), Eint 1,
            Times (Val (Ide "x"),
                Appl (Val (Ide "fact"), Diff (Val (Ide "x"), Eint 1))))),
        Diff (Val (Ide "x"), Eint 1))),
    <fun>)

```

```

Let(Ide "f", Fun(Ide "x",Fun(Ide "y",Val(Ide "x"))),
    Appl(Appl(Val(Ide "f"),Eint 3),Echar 'c'))

```

```

tipo: etype = TInt
valore: eval = Int 3

```

```

Fun(Ide "x",Fun(Ide "y",Val(Ide "x")))

```

```

tipo: etype = TFun (TVar "?T01", TFun (TVar "?T02", TVar "?T01"))
valore: eval = Closure (Fun (Ide "x", Fun (Ide "y", Val (Ide "x"))),
    <fun>)

```

```

Cons (Empty, Empty)

```

```

tipo: etype = TList [TList [TVar "?T01"]]
valore: eval = List [List []]

```

```

Cons(Eint 1, Cons(Eint 2, Empty))

```

```

tipo: etype = TList [TInt]
valore: eval = List [Int 1; Int 2]

```

```

Eq(Cons(Eint 1, Cons(Eint 2, Empty)),
    Cons(Eint 1, Cons(Eint 2, Empty)))

```

```

tipo: etype = TBool
valore: eval = Bool true

```

```

Eq(Eq(Cons(Eint 1, Cons(Eint 2, Empty)),
    Cons(Eint 1, Cons(Eint 2, Empty))),
    False)

```

```

tipo: etype = TBool
valore: eval = Bool false

```

```
Fun(Ide "x", Ifthenelse(Eq(Val(Ide "x"),Empty),True,False))
```

```
tipo: etype = TFun (TList [TVar "?T30"], TBool)
```

```
valore: eval =
```

```
Closure (Fun (Ide "x", Ifthenelse (Eq (Val (Ide "x")), Empty), True,
False)),
  <fun>)
```

```
Let(Ide "f",Fun(Ide "x", Ifthenelse(Eq(Val(Ide
"x"),Empty),True,False)),Appl(Val(Ide "f"),Cons(Eint 2, Empty)))
```

```
tipo: etype = TBool
```

```
valore: eval = Bool false
```

```
Cons(Cons(Eint 1,Empty),Empty)
```

```
tipo: etype = TList [TList [TInt]]
```

```
valore: eval = List [List [Int 1]]
```

```
Epair(Fun(Ide "x", Ifthenelse(Eq(Val(Ide "x"),Empty),True,False)),
  Cons(Cons(Eint 1,Empty),Empty))
```

```
tipo: etype = TPair (TFun (TList [TVar "?T43"], TBool), TList [TList
[TInt]])
```

```
valore: eval =
```

```
Pair
```

```
(Closure
  (Fun (Ide "x", Ifthenelse (Eq (Val (Ide "x")), Empty), True,
False)),
  <fun>),
  List [List [Int 1]])
```

```
Appl(Fst(Epair(Fun(Ide "x", Ifthenelse(Eq(Val(Ide "x"),Empty),
True,
False))),
  Cons(Cons(Eint 1,Empty),Empty))),
  Snd(Epair(Fun(Ide "x", Ifthenelse(Eq(Val(Ide "x"),Empty),
True,
False))),
  Cons(Cons(Eint 1,Empty),Empty)))
```

```
tipo: etype = TBool
```

```
valore: eval = Bool false
```

```

Let(Ide "p",Epair(Fun(Ide "x", Ifthenelse(Eq(Val(Ide "x"),Empty),
                                         True,
                                         False))),
      Cons(Cons(Eint 1,Empty),Empty)),
  Appl(Fst(Val(Ide "p")),Snd(Val(Ide "p"))))

```

tipo: `etype = TBool`
 valore: `eval = Bool false`

```

Cons(Eint 1,Cons(True, Empty))

```

tipo: non lo deve trovare. Non è tipabile

valore: avete due possibilità. Una è trovare che il valore è `eval = List [Int 1; Bool true]`, la seconda che non ha valore dato che i tipi non sono corretti. Dipende da come avete implementato il tipaggio nell'interprete. La prima soluzione però non è bella. Cercate quindi di non averla, dato che comporta una penalizzazione.

```

Eq(Cons(Eint 1,Empty),Cons(True,Empty))

```

tipo: anche questo non è tipabile, dato che l'uguale deve avere due espressioni dello stesso tipo

valore: avete due possibilità. Una è trovare che il valore è `eval = Bool false`, la seconda che non ha valore dato che i tipi non sono corretti. Dipende da come avete implementato il tipaggio nell'interprete. Entrambe le soluzioni sono ugualmente accettate

```

Let(Ide "f",Rec(Ide "f",Fun(Ide "x",Ifthenelse(
  Eq(Val(Ide "x"),Eint 0),
  Empty,
  Cons(Val(Ide "x"),Appl(Val(Ide "f"),Diff(Val(Ide "x"),Eint 1)))))),
  Appl(Val(Ide "f"),Eint 5))

```

tipo: `etype = TList [TInt]`
 valore: `eval = List [Int 5; Int 4; Int 3; Int 2; Int 1]`