as04

By Martin Lundfall, Denis Erfurt

May 17, 2016

Contents

```
theory LundfallErfurtEx04
imports Main
begin




lemma fromEx2:
  assumes 1: ¬A
  shows A ⟶ B
  proof -
    {
      assume 2: A
      {
        assume 3: ¬B
        from 1 2 have False by (rule notE)
      }
      from this have B by (rule ccontr)
    }
    from this have A ⟶ B by (rule impI)
  thus ?thesis .
qed

theorem ontological:
assumes 1: ¬G ⟶ ¬(P ⟶ A) and
2: ¬P
shows G
proof -
  {
    assume 3: ¬G
    from 1 3 have 4: ¬(P ⟶ A) by (rule mp)
    from 2 have 5: P ⟶ A by (rule fromEx2)
    from 4 5 have False by (rule notE)
  }
  from this have G by (rule ccontr)
```

thus *?thesis* **.**
qed

One argument could be that we do not accept the law of double negation. Then we cannot conclude the existence of god from refuting the non-existence of god. Also, the way the implication is interpreted in classical logic is not they way we necessarily use it in everyday language. The assumption 'it is not that case that (if I pray, my prayers will be answered)' is different from 'if I pray, my prayers will not be answered', which is probably a more reasonable assumption in the context of gods nonexistence.

Ex 2

```
fun sum-n :: nat ⇒ nat where
  sum-n 0 = 0 |
  sum-n (Suc n) = Suc n + sum-n n

lemma sum-n n = n * (n + 1) div 2

proof (induction n)
  case 0
  show ?case by simp
next
  case (Suc n)
  from this have sum-n (Suc n) = (Suc n) * ((Suc n) + 1) div 2 by simp
  thus ?case .
qed

fun sum-n-square :: nat ⇒ nat where
  sum-n-square 0 = 0 |
  sum-n-square (Suc n) = Suc n * Suc n + sum-n-square n

lemma sum-n-square n = (n * (n + 1) * (2 * n + 1)) div 6
proof (induction n)
  case 0
  show ?case by simp
next
  case (Suc n)
  from this have sum-n-square (Suc n) = Suc n * Suc n + sum-n-square n by
simp
  then have sum-n-square (Suc n) = Suc n * Suc n + (n * (n + 1) * (2 * n +
1)) div 6 by (simp add: Suc.IH)
  then have sum-n-square (Suc n) = ((Suc n) * (n + 1) * 6 + (Suc n) * n * (2
* n + 1)) div 6 by simp
  then have sum-n-square (Suc n) = ((Suc n) * ((n + 1) * 6) + (Suc n) * n *
(2 * n + 1)) div 6 using mult.assoc [of Suc n n + 1 6] by simp
  then have sum-n-square (Suc n) = ((Suc n) * ((n + 1) * 6) + (Suc n) * (n *
(2 * n + 1))) div 6 using mult.assoc [of Suc n n 2 * n + 1] by simp
  then have sum-n-square (Suc n) = ((Suc n) * (((n + 1) * 6) + (n * (2 * n +
1)))) div 6 using add-mult-distrib2 [of Suc n (n + 1) * 6  n * (2 * n + 1)] by
```

simp
 then have *sum-n-square* (*Suc n*) = ((*Suc n*) ∗ ((*2* ∗ *n* + *3*) ∗ *2* + *2* ∗ *n* + *n* ∗ (*2* ∗ *n* + *1*))) *div 6* **by** *simp*
 then have *sum-n-square* (*Suc n*) = ((*Suc n*) ∗ ((*2* ∗ *n* + *3*) ∗ *2* + *n* ∗ (*2* ∗ *n* + *3*))) *div 6* **using** *add-mult-distrib2* **by** *simp*
 then have *sum-n-square* (*Suc n*) = ((*Suc n*) ∗ (*2* ∗ *n* + *3*) ∗ (*n* + *2*)) *div 6* **by** (*simp add*: *add-mult-distrib2 mult.commute*)
 then have *sum-n-square* (*Suc n*) = (*Suc n* ∗ (*2* ∗ *n* + *3*) ∗ (*Suc n* + *1*)) *div 6* **by** (*simp add*: *add-mult-distrib2 mult.commute*)
 then have *sum-n-square* (*Suc n*) = (*Suc n* ∗ ((*2* ∗ *n* + *3*) ∗ (*Suc n* + *1*))) *div 6* **by** (*simp add*: *add-mult-distrib2 mult.commute*)
 then have *sum-n-square* (*Suc n*) = (*Suc n* ∗ ((*Suc n* + *1*) ∗ (*2* ∗ *n* + *3*))) *div 6* **by** (*simp add*: *mult.commute*)
 then have *sum-n-square* (*Suc n*) = (*Suc n* ∗ ((*Suc n* + *1*) ∗ (*2* ∗ *n* + (*2* ∗ *1* + *1*)))) *div 6* **by** (*simp add*: *mult.assoc* [*symmetric*])
 then have *sum-n-square* (*Suc n*) = (*Suc n* ∗ ((*Suc n* + *1*) ∗ (*2* ∗ *Suc n* + *1*))) *div 6* **by** (*simp add*: *add-mult-distrib2*)
 then have *sum-n-square* (*Suc n*) = (*Suc n* ∗ (*Suc n* + *1*) ∗ (*2* ∗ *Suc n* + *1*)) *div 6* **using** *mult.assoc* [*of Suc n Suc n* + *1 2* ∗ *Suc n* + *1*] **by** *simp*
thus *?case* **.**
qed

lemma *flipping*:
 assumes *A* ⟶ *B*
 shows ¬ *B* ⟶ ¬ *A*
proof −
 {
  assume *1*: ¬*B*
  {
   assume *2*: *A*
   from *assms 2* **have** *3*: *B* **by** (*rule mp*)
   from *1 3* **have** *False* **by** (*rule notE*)
  }
  from *this* **have** ¬*A* **by** (*rule notI*)
 }
 from *this* **have** ¬ *B* ⟶ ¬ *A* **by** (*rule impI*)
thus *?thesis* **.**
qed

theorem *Ex3*:
assumes *1*: ∀ *X*. ¬*rich*(*X*) ⟶ *rich* (*parent*(*X*))
shows ∃ *X*. *rich*(*parent*(*parent*(*X*)))∧*rich*(*X*)
proof *cases*
 assume *2*: ∀ *X*. ¬*rich*(*X*)
  {
   assume ¬(∃ *X*. *rich*(*parent*(*parent*(*X*))) ∧ *rich*(*X*))
   fix *X*
   from *2* **have** *3*: ¬*rich*(*X*) **by** (*rule allE*)
   from *1* **have** *4*: ¬*rich*(*X*) ⟶ *rich*(*parent*(*X*)) **by** (*rule allE*)

```
      from 4 3 have 5: rich(parent(X)) by (rule mp)
      then have 6: ∃ X. rich(X) by (rule exI)
      from 2 have 7: ¬ (∃  X. rich(X)) by simp
      from 7 6 have False by (rule notE)
    }
  from this show ∃ X. rich(parent(parent(X)))∧rich(X) by (rule ccontr)
next
  assume 8: ¬(∀ X. ¬rich(X))
  from 8 have ∃ X. rich(X) by simp
  then obtain y where 9: rich(y) by (rule exE)
  {
    assume 11: ¬rich(parent(parent(y)))
    {
      fix x
      from assms have ¬rich(x) ⟶ rich(parent(x)) by (rule allE)
      then have ¬rich(parent(x)) ⟶ ¬(¬rich(x)) by (rule flipping)
      then have ¬rich(parent(x)) ⟶ rich(x) by simp
    }
    from this have 12: ∀ x. ¬rich(parent(x)) ⟶ rich(x) by (rule allI)
    from 11 12 have 13: rich(parent(y)) by simp
    from 1 11 have 14: rich(parent(parent(parent(y)))) by simp
    from 13 14 have rich(parent(parent(parent(y)))) ∧ rich(parent(y)) by simp
    then have ∃ y. rich(parent(parent(y)))∧rich(y) by (rule exI)
    } note case1 = this
    {
      assume 15: rich(parent(parent(y)))
      from 9 15 have rich(parent(parent(y))) ∧ rich(y) by simp
      then have ∃ y. rich(parent(parent(y)))∧rich(y) by (rule exI)
    } note case2 = this
    from case1 case2 show ∃ y. rich(parent(parent(y))) ∧ rich(y) by cases
qed
end
```