



Theorembeweiserpraktikum – SS 2016

<http://pp.ipd.kit.edu/lehre/SS2016/tba>

Lösung 2: Simplifikation

Abgabe: 2. Mai 2016, 12:00 Uhr
Besprechung: 3. Mai 2016

1 Prädikatenlogik

Es geht wieder um Beweise mit den Regeln des Kalküls des natürlichen Schließens. Zusätzlich zu den Regeln der letzten Übung können Sie nun auch folgende Regeln verwenden:

$allI: (\bigwedge x. P x) \implies \forall x. P x$ $allE: \forall x. P x \implies (P x \implies R) \implies R$

$exI: P x \implies \exists x. P x$ $exE: \exists x. P x \implies (\bigwedge x. P x \implies Q) \implies Q$

Es dürfen wieder nur die Befehle **proof** (mit (*rule ...*)), **assume**, **have**, **show**, **next**, **qed** und **from** sowie die darauf aufbauenden Abkürzungen verwendet werden. Zusätzlich dürfen die Befehle **fix** und **obtain** verwendet werden.

Beispiel

lemma " $\forall x. P x \longrightarrow (\exists x. P x)$ "

proof(*rule allI*)

fix x

show " $P x \longrightarrow (\exists x. P x)$ "

proof

assume " $P x$ "

then show " $\exists x. P x$ " **by** (*rule exI*)

qed

qed

lemma " $(\forall x. P x) \longleftrightarrow \neg (\exists x. \neg P x)$ "

proof

assume " $\forall x. P x$ "

show " $\neg (\exists x. \neg P x)$ "

proof

assume " $\exists x. \neg P x$ "

then obtain x **where** " $\neg P x$ " **by** (*rule exE*)

from $\langle \forall x. P x \rangle$

have " $P x$ " **by** (*rule allE*)

with $\langle \neg P x \rangle$

show *False..*

qed

next

assume *lhs*: " $\neg (\exists x. \neg P x)$ "

show " $\forall x. P x$ "

proof(*rule allI*)

```

fix x
show "P x"
proof(rule ccontr)
  assume "¬ P x"
  then have "∃x. ¬ P x" by (rule exI)
  with lhs
  show False..
qed
qed
qed

lemma "(¬ (∀x. P x)) ↔ (∃x. ¬ P x)"
proof
  assume "¬ (∀x. P x)"
  show "∃x. ¬ P x"
  proof(rule ccontr) — Regel contrapos_np: ¬ A ⇒ (¬ B ⇒ A) ⇒ B wäre kürzer
    assume "¬ (∃x. ¬ P x)"
    have "∀ x. P x"
    proof
      fix x
      show "P x"
      proof(rule ccontr)
        assume "¬ P x"
        then have "∃x. ¬ P x"..
        with ⟨¬ ?this⟩ — ?this ist die letzte Aussage, also ∃x. ¬ P x
        show False..
      qed
    qed
    with ⟨¬ ?this⟩
    show False..
  qed
next
  assume "∃x. ¬ P x"
  then obtain x where "¬ P x"..
  show "¬ (∀x. P x)"
  proof
    assume "∀x. P x"
    then have "P x"..
    with ⟨¬ ?this⟩
    show False..
  qed
qed

lemma "(∀x. P x → Q) ↔ ((∃x. P x) → Q)"
proof
  assume "∀x. P x → Q"
  show "(∃x. P x) → Q"
  proof
    assume "∃x. P x"
    then obtain x where "P x"..
    from ⟨∀x. P x → Q⟩ have "P x → Q"..
    from this ⟨P x⟩

```

```

    show  $Q$ ..
  qed
next
  assume " $(\exists x. P x) \longrightarrow Q$ "
  show " $\forall x. P x \longrightarrow Q$ "
  proof
    fix  $x$ 
    show " $P x \longrightarrow Q$ "
    proof
      assume " $P x$ "
      then have " $\exists x. P x$ "..
      with  $\langle (\exists x. P x) \longrightarrow Q \rangle$ 
      show  $Q$ ..
    qed
  qed
qed

```

```

lemma " $(\exists x. \forall y. P x y) \longrightarrow (\forall y. \exists x. P x y)$ "
proof
  assume " $\exists x. \forall y. P x y$ "
  then obtain  $x$  where " $\forall y. P x y$ "..
  show " $\forall y. \exists x. P x y$ "
  proof
    fix  $y$ 
    from  $\langle \forall y. P x y \rangle$ 
    have " $P x y$ "..
    then show " $\exists x. P x y$ "..
  qed
qed

```

```

lemma " $(\forall x. P x) \wedge (\forall x. Q x) \longleftrightarrow (\forall x. (P x \wedge Q x))$ "
proof
  assume " $(\forall x. P x) \wedge (\forall x. Q x)$ "
  then have " $\forall x. P x$ "..
  from  $\langle (\forall x. P x) \wedge (\forall x. Q x) \rangle$ 
  have " $\forall x. Q x$ "..

  show " $\forall x. (P x \wedge Q x)$ "
  proof
    fix  $x$ 
    show " $P x \wedge Q x$ "
    proof
      from  $\langle \forall x. P x \rangle$  show " $P x$ "..
    next
      from  $\langle \forall x. Q x \rangle$  show " $Q x$ "..
    qed
  qed
next
  assume " $\forall x. (P x \wedge Q x)$ "
  show " $(\forall x. P x) \wedge (\forall x. Q x)$ "
  proof
    show " $\forall x. P x$ "
  qed

```

```

proof
  fix x
  from  $\langle \forall x. (P\ x \wedge Q\ x) \rangle$ 
  have "P x  $\wedge$  Q x"..
  then show "P x"..
qed
next
show " $\forall x. Q\ x$ "
proof
  fix x
  from  $\langle \forall x. (P\ x \wedge Q\ x) \rangle$ 
  have "P x  $\wedge$  Q x"..
  then show "Q x"..
qed
qed
qed

lemma " $(\exists x. P\ x) \vee (\exists x. Q\ x) \longleftrightarrow (\exists x. (P\ x \vee Q\ x))$ "
proof
assume " $(\exists x. P\ x) \vee (\exists x. Q\ x)$ "
then show " $\exists x. (P\ x \vee Q\ x)$ "
proof
  assume " $\exists x. P\ x$ "
  then obtain x where "P x"..
  then have "P x  $\vee$  Q x"..
  then show " $\exists x. (P\ x \vee Q\ x)$ "..
next
  assume " $\exists x. Q\ x$ "
  then obtain x where "Q x"..
  then have "P x  $\vee$  Q x"..
  then show " $\exists x. (P\ x \vee Q\ x)$ "..
qed
next
assume " $\exists x. (P\ x \vee Q\ x)$ "
then obtain x where "P x  $\vee$  Q x"..
then show " $(\exists x. P\ x) \vee (\exists x. Q\ x)$ "
proof
  assume "P x"
  then have " $\exists x. P\ x$ "..
  then show " $(\exists x. P\ x) \vee (\exists x. Q\ x)$ "..
next
  assume "Q x"
  then have " $\exists x. Q\ x$ "..
  then show " $(\exists x. P\ x) \vee (\exists x. Q\ x)$ "..
qed
qed

```

Bei diesem Lemma darf mit Fallunterscheidung (Methode *cases*) gearbeitet werden. Erinnerung: Eine Variable, über die Sie nichts wissen (brauchen), erhalten Sie mit **fix**.

```

lemma " $\exists x. P\ x \longrightarrow (\forall x. P\ x)$ "
proof (cases " $\exists x. \neg P\ x$ ")
  assume " $\exists x. \neg (P\ x)$ "

```

```

then obtain  $x$  where " $\neg P x$ "..
have " $P x \longrightarrow (\forall x. P x)$ "
proof
  assume " $P x$ "
  with  $\langle \neg P x \rangle$ 
  show " $\forall x. P x$ "..
qed
then show " $\exists x. P x \longrightarrow (\forall x. P x)$ "..
next
assume " $\neg (\exists x. \neg P x)$ "
have " $\forall x. P x$ "
proof
  fix  $x$ 
  show " $P x$ "
  proof(rule ccontr)
    assume " $\neg P x$ "
    then have " $\exists x. \neg P x$ "..
    with  $\langle \neg ?this \rangle$ 
    show False..
  qed
qed

fix  $x$ 
from  $\langle \forall x. P x \rangle$ 
have " $P x \longrightarrow (\forall x. P x)$ "..
then show " $\exists x. P x \longrightarrow (\forall x. P x)$ "..
qed

```

2 Definitionen und Arbeiten mit Gleichheit

In klassischer Aussagenlogik können alle Aussagen allein aus *False* und *nor* gebildet werden. Definieren Sie den *nor*-Operator:

definition

```

nor :: "bool  $\Rightarrow$  bool  $\Rightarrow$  bool" (infix " $\downarrow$ " 37)
where " $A \downarrow B \longleftrightarrow \neg (A \vee B)$ "

```

Nun leiten Sie Lemmas her, die die üblichen booleschen Junktoren nur mit *False* und *op* \downarrow darstellen. Theoretisch ist *op* \downarrow alleine schon universell. Der Einfachheit halber dürfen Sie in dieser Aufgabe aber auch noch *False* zusätzlich verwenden. Gehen Sie dabei wie folgt vor:

- Wenden Sie zu Beginn keine Regel an (**proof-**).
- Schreiben Sie erst mit **have** $.... =$ den Ausdruck in eine Form um, die neben Ausdrücken der Form $\neg (.... \vee)$ nur Junktoren verwendet, für die Sie bereits Regeln geschrieben haben.
- Führen Sie diese dann Schrittweise mit **also have** in die gewünschte Form über. Dabei sollten Sie neben *nor_def[symmetric]* nur die davor bewiesenen Regeln *rewrite_foo* in ihren **from**-Befehlen aufführen müssen, und als Beweis dann **.** oder **by** (*rule arg_cong*) verwenden.

- Wenn die Gleichungskette zum Lemma passt, lässt sie sich mit **finally show ?thesis** . abschließen.

Um *True* zu zeigen verwenden Sie die Regel *TrueI: True*.

```
lemma rewrite_not: "¬ A ↔ False ↓ A"
proof-
  have "¬ A ↔ ¬ (False ∨ A)"
  proof
    assume "¬ A"
    show "¬ (False ∨ A)"
    proof
      assume "False ∨ A"
      then show False
      proof
        assume A
        with ⟨¬A⟩
        show False..
      qed
    qed
  next
  assume "¬ (False ∨ A)"
  show "¬ A"
  proof
    assume A
    then have "False ∨ A"..
    with ⟨¬ ?this⟩
    show False..
  qed
qed
also
from nor_def[symmetric]
have "... ↔ False ↓ A".
finally
show ?thesis.
qed
```

```
lemma rewrite_or: "A ∨ B ↔ False ↓ (A ↓ B)"
proof-
  have "A ∨ B ↔ ¬ (¬ (A ∨ B))"
  proof
    assume "A ∨ B"
    show "¬ (¬ (A ∨ B))"
    proof
      assume "¬ (A ∨ B)"
      from this ⟨A ∨ B⟩
      show False..
    qed
  next
  assume "¬ (¬ (A ∨ B))"
  show "A ∨ B"
  proof(rule ccontr)
```

```

    assume "¬ (A ∨ B)"
    with ⟨¬ (¬ (A ∨ B))⟩
    show False..
  qed
qed
also
from nor_def[symmetric]
have "... ⟷ ¬ (A ↓ B)" by (rule arg_cong)
also
from rewrite_not
have "... ⟷ False ↓ (A ↓ B)".
finally
show "A ∨ B ⟷ False ↓ (A ↓ B)".
qed

lemma rewrite_and: "A ∧ B ⟷ (False ↓ A) ↓ (False ↓ B)"
proof-
  have "A ∧ B ⟷ ¬ (¬ A ∨ ¬ B)"
  proof
    assume "A ∧ B" then have A..
    from ⟨A ∧ B⟩ have B..
    show "¬ (¬ A ∨ ¬ B)"
    proof
      assume "¬ A ∨ ¬ B"
      then show False
      proof
        assume "¬ A"
        from this ⟨A⟩
        show False..
      next
        assume "¬ B"
        from this ⟨B⟩
        show False..
      qed
    qed
  next
  assume "¬ (¬ A ∨ ¬ B)"
  show "A ∧ B"
  proof
    show A
    proof (rule ccontr)
      assume "¬ A"
      then have "¬ A ∨ ¬ B"..
      with ⟨¬ (¬ A ∨ ¬ B)⟩
      show False..
    qed
  next
  show B
  proof (rule ccontr)
    assume "¬ B"
    then have "¬ A ∨ ¬ B"..
    with ⟨¬ (¬ A ∨ ¬ B)⟩

```

```

    show False..
  qed
qed
qed
also
from nor_def[symmetric]
have "...  $\longleftrightarrow (\neg A) \downarrow (\neg B)$ ".
also
from rewrite_not
have "...  $\longleftrightarrow (\text{False} \downarrow A) \downarrow (\neg B)$ " by (rule arg_cong)
also
from rewrite_not
have "...  $\longleftrightarrow (\text{False} \downarrow A) \downarrow (\text{False} \downarrow B)$ " by (rule arg_cong)
finally
show " $A \wedge B \longleftrightarrow (\text{False} \downarrow A) \downarrow (\text{False} \downarrow B)$ ".
qed

lemma rewrite_imp: " $(A \longrightarrow B) \longleftrightarrow \text{False} \downarrow ((\text{False} \downarrow A) \downarrow B)$ "
proof-
  have " $(A \longrightarrow B) \longleftrightarrow \neg A \vee B$ "
  proof
    assume "A  $\longrightarrow$  B"
    show " $\neg A \vee B$ "
    proof(cases A)
      assume " $\neg A$ "
      then show " $\neg A \vee B$ "..
    next
      assume A
      with  $\langle A \longrightarrow B \rangle$ 
      have B..
      then show " $\neg A \vee B$ "..
    qed
  next
    assume " $\neg A \vee B$ "
    then show " $A \longrightarrow B$ "
    proof
      assume " $\neg A$ "
      show " $A \longrightarrow B$ "
      proof
        assume A
        with  $\langle \neg A \rangle$ 
        show B..
      qed
    next
      assume B
      then show " $A \longrightarrow B$ "..
    qed
  qed
also
from rewrite_or
have "...  $\longleftrightarrow \text{False} \downarrow ((\neg A) \downarrow B)$ ".
also

```



```

from rewrite_not
have "...  $\longleftrightarrow$  False  $\downarrow$  ((False  $\downarrow$  A)  $\downarrow$  B)" by (rule arg_cong)
finally
show "(A  $\longrightarrow$  B)  $\longleftrightarrow$  False  $\downarrow$  ((False  $\downarrow$  A)  $\downarrow$  B)".
qed

```

```

lemma rewrite_True: "True  $\longleftrightarrow$  False  $\downarrow$  False"
proof-
have "True  $\longleftrightarrow$   $\neg$  False"
proof
show True by (rule TrueI)
next
show " $\neg$  False" by (rule notI)
qed
also from rewrite_not have "...  $\longleftrightarrow$  False  $\downarrow$  False" .
finally show ?thesis.
qed

```

Schreiben Sie nun den folgenden Ausdruck schrittweise so um, das er nur mit *op* \downarrow und *False* gebildet wird. Verwenden Sie dabei das **also ... finally** Konstrukt.

```

lemma "(A  $\longrightarrow$  (A  $\vee$  B))  $\wedge$   $\neg$  B  $\longleftrightarrow$ 
(False  $\downarrow$  (False  $\downarrow$  ((False  $\downarrow$  A)  $\downarrow$  (False  $\downarrow$  (A  $\downarrow$  B))))  $\downarrow$  (False  $\downarrow$  (False  $\downarrow$  B))"
proof-
from rewrite_and
have "(A  $\longrightarrow$  (A  $\vee$  B))  $\wedge$   $\neg$  B  $\longleftrightarrow$  (False  $\downarrow$  (A  $\longrightarrow$  (A  $\vee$  B)))  $\downarrow$  (False  $\downarrow$  ( $\neg$  B))".
also
from rewrite_not
have "...  $\longleftrightarrow$  (False  $\downarrow$  (A  $\longrightarrow$  (A  $\vee$  B)))  $\downarrow$  (False  $\downarrow$  (False  $\downarrow$  B))"
by (rule arg_cong)
also
from rewrite_imp
have "...  $\longleftrightarrow$  (False  $\downarrow$  (False  $\downarrow$  ((False  $\downarrow$  A)  $\downarrow$  (A  $\vee$  B))))  $\downarrow$  (False  $\downarrow$  (False  $\downarrow$  B))"
by (rule arg_cong)
also
from rewrite_or
have "...  $\longleftrightarrow$  (False  $\downarrow$  (False  $\downarrow$  ((False  $\downarrow$  A)  $\downarrow$  (False  $\downarrow$  (A  $\downarrow$  B))))  $\downarrow$  (False  $\downarrow$  (False
 $\downarrow$  B))"
by (rule arg_cong)
finally
show ?thesis.
qed

```