Karlsruher Institut für Technologie

Institut für Programmstrukturen und Datenorganisation
Lehrstuhl Programmierparadigmen
Am Fasanengarten 5
76131 Karlsruhe
http://pp.ipd.kit.edu/

Theorembeweiserpraktikum – SS 2018

http://pp.ipd.kit.edu/lehre/SS2018/tba

Lösung 1: Deduktion

Abgabe: 23. April 2018, 12:00 Uhr
Besprechung: 24. April 2018

1 Natürliches Schließen

In dieser Aufgabe geht es um den Kalkül des natürlichen Schließens, mit dessen Hilfe einige Lemmas der Aussagen-Logik bewiesen werden sollen (nächste Seite).
Für die Beweise gelten die folgenden Spielregeln:

- Es dürfen nur die Befehle **proof**, **assume**, **have**, **show**, **next**, **qed**, und **from** verwendet werden, sowie darauf aufbauende Abkürzungen wie **then**, **with**, **..** und **.**.
- Der Befehl **proof** darf nur als **proof** (*rule regel*) (oder als **proof** -) verwendet werden, wobei die Regel eine der folgenden ist: (Anzeigen der Lemmas mittel **thm** *lemma-Name*)

  impI: $(A \Longrightarrow B) \Longrightarrow A \longrightarrow B$, *impE:* $A \longrightarrow B \Longrightarrow A \Longrightarrow (B \Longrightarrow C) \Longrightarrow C$,
  conjI: $A \Longrightarrow B \Longrightarrow A \wedge B$, *conjE:* $A \wedge B \Longrightarrow (A \Longrightarrow B \Longrightarrow C) \Longrightarrow C$,
  disjI1: $A \Longrightarrow A \vee B$, *disjE:*
  disjI2: $B \Longrightarrow A \vee B$, $A \vee B \Longrightarrow (A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C$,
  notI: $(A \Longrightarrow False) \Longrightarrow \neg A$, *notE:* $\neg A \Longrightarrow A \Longrightarrow B$,
  iffI: $(A \Longrightarrow B) \Longrightarrow (B \Longrightarrow A) \Longrightarrow A \longleftrightarrow B$,
  iffE: $A \longleftrightarrow B \Longrightarrow (A \longrightarrow B \Longrightarrow B \longrightarrow A \Longrightarrow C) \Longrightarrow C$,
  ccontr: $(\neg A \Longrightarrow False) \Longrightarrow A$
  classical: $(\neg A \Longrightarrow A) \Longrightarrow A$

  Alle diese Regeln, außer den letzten beiden, sind als Standard-Regeln vorgegeben, das heißt der Befehl **proof** (*rule*) (oder kurz **proof**) wählt die passende Regel aus, auch ohne dass man sie explizit angibt. Lassen Sie nur Namen von Regeln weg, die sie zuvor zumindest einmal explizit verwendet haben.

Beispiel

```
lemma imp_uncurry: "(P ⟶ (Q ⟶ R)) ⟶ P ∧ Q ⟶ R"
proof (rule impI)
  assume PQR: "P ⟶ (Q ⟶ R)"
  show "P ∧ Q ⟶ R"
  proof — Das (rule impI) kann wegelassen werden
    assume "P ∧ Q"
    then have "P" by (rule conjE)
    with PQR
    have "Q ⟶ R" by (rule impE)
```

```
    from ‹P ∧ Q›
    have "Q".. — Hier steht eigentlich by (rule conjE)
    with ‹Q ⟶ R›
    show R..
  qed
qed

lemma I: "A ⟶ A"
by (rule impI)

lemma "A ∧ B ⟶ B ∧ A"
proof
  assume "A ∧ B"
  then have "A" by (rule conjE)

  from ‹A ∧ B›
  have "B"..

  from ‹B› ‹A›
  show "B ∧ A" by (rule conjI)
qed

lemma "A ∧ B ⟶ A ∨ B"
proof
  assume "A ∧ B"
  then have "A"..
  then show "A ∨ B" by (rule disjI1)
qed

lemma "((A ∨ B) ∨ C) ⟶ A ∨ (B ∨ C)"
proof
  assume "(A ∨ B) ∨ C"
  then show "A ∨ (B ∨ C)"
  proof (rule disjE)
    assume "A ∨ B"
    then show "A ∨ (B ∨ C)"
    proof
      assume A
      then show "A ∨ (B ∨ C)" by (rule disjI1)
    next
      assume B
      then have "B ∨ C"..
      then show "A ∨ (B ∨ C)" by (rule disjI2)
    qed
  next
    assume C
    then have "B ∨ C"..
    then show "A ∨ (B ∨ C)"..
  qed
qed

lemma K: "A ⟶ B ⟶ A"
```

```
proof
  assume "A"
  show "B ⟶ A"
  proof
    from ‹A›
    show "A".
  qed
qed

lemma "A ∨ A ⟷ A ∧ A"
proof
  assume "A ∨ A"
  then show "A ∧ A"
  proof
    assume A
    from ‹A› ‹A›
    show "A ∧ A"..
  next
    assume A
    from ‹A› ‹A›
    show "A ∧ A"..
  qed
next
  assume "A ∧ A"
  then have "A"..
  then show "A ∨ A"..
qed

lemma S: "(A ⟶ B ⟶ C) ⟶ (A ⟶ B) ⟶ A ⟶ C"
proof
  assume ABC: "A ⟶ B ⟶ C"
  show "(A ⟶ B) ⟶ A ⟶ C"
  proof
    assume "A ⟶ B"
    show "A ⟶ C"
    proof
      assume "A"
      with ‹A ⟶ B›
      have "B" by (rule impE)

      from ABC ‹A›
      have "B ⟶ C"..

      from ‹B ⟶ C› ‹B›
      show "C"..
    qed
  qed
qed

lemma "(A ⟶ B) ⟶ (B ⟶ C) ⟶ A ⟶ C"
proof
  assume "A ⟶ B"
```

```
  show "(B ⟶ C) ⟶ A ⟶ C"
  proof
    assume "B ⟶ C"
    show "A ⟶ C"
    proof
      assume "A"
      with ‹A ⟶ B›
      have B ..
      with ‹B ⟶ C›
      show C ..
    qed
  qed
qed

lemma "¬ ¬ A ⟶ A"
proof
  assume "¬ ¬ A"
  show A
  proof (rule ccontr)
    assume "¬A"
    with ‹¬ ¬ A›
    show False by (rule notE)
  qed
qed

lemma "A ⟶ ¬ ¬ A"
proof
  assume A
  show "¬ ¬ A"
  proof (rule notI)
    assume "¬ A"
    from this ‹A›
    show False ..
  qed
qed

lemma "(¬ A ⟶ B) ⟶ (¬ B ⟶ A)"
proof
  assume "¬ A ⟶ B"
  show "¬ B ⟶ A"
  proof
    assume "¬ B"
    show A
    proof (rule ccontr)
      assume "¬A"
      with ‹¬ A ⟶ B›
      have "B"..
      with ‹¬ B›
      show False..
    qed
  qed
qed
```

```
lemma "((A ⟶ B) ⟶ A) ⟶ A"
proof
  assume ABA: "((A ⟶ B) ⟶ A)"
  show A
  proof (rule classical)
    assume "¬A"

    have "A ⟶ B"
    proof
      assume A
      with ‹¬ A›
      show B..
    qed
    with ABA
    show A..
  qed
qed

lemma "A ∨ ¬ A"
proof(rule classical)
  assume "¬ (A ∨ ¬ A)"

  have "¬A"
  proof
    assume A
    then have "A ∨ ¬ A" ..
    with ‹¬ (A ∨ ¬ A)›
    show False..
  qed
  then show "A ∨ ¬A"..
qed

lemma deMorgan1: "¬ (A ∨ B) ⟷ ¬ A ∧ ¬ B"
proof(rule iffI)
  assume "¬ (A ∨ B)"
  show "¬ A ∧ ¬ B"
  proof
    show "¬A"
    proof
      assume "A"
      then have "A ∨ B"..
      with ‹¬ (A ∨ B)›
      show False ..
    qed
  next
    show "¬B"
    proof
      assume "B"
      then have "A ∨ B"..
      with ‹¬ (A ∨ B)›
      show False ..
    qed
```

```
  qed
next
  assume "¬ A ∧ ¬ B"
  then have "¬A"..

  from ‹¬ A ∧ ¬ B›
  have "¬ B"..

  show "¬ (A ∨ B)"
  proof
    assume "A ∨ B"
    then show False
    proof
      assume "A"
      with ‹¬A›
      show False..
    next
      assume "B"
      with ‹¬B›
      show False..
    qed
  qed
qed

lemma deMorgan2: "¬ (A ∧ B) ⟷ ¬ A ∨ ¬ B"
proof
  assume "¬ (A ∧ B)"
  show "¬ A ∨ ¬ B"
  proof (rule classical)
    assume "¬ (¬ A ∨ ¬ B)"
    have "A"
    proof (rule ccontr)
      assume "¬A"
      then have "¬ A ∨ ¬ B"..
      with ‹¬ (¬ A ∨ ¬ B)›
      show False..
    qed

    have "B"
    proof (rule ccontr)
      assume "¬B"
      then have "¬ A ∨ ¬ B"..
      with ‹¬ (¬ A ∨ ¬ B)›
      show False..
    qed

    from ‹A› ‹B› have "A ∧ B"..
    with ‹¬ (A ∧ B)›
    show ?thesis..
  qed
next
  assume "¬ A ∨ ¬ B"
```

```
  then show "¬ (A ∧ B)"
  proof
    assume "¬ A"
    show ?thesis
    proof
      assume "A ∧ B"
      then have "A"..
      with ‹¬A›
      show False..
    qed
  next
    assume "¬ B"
    show ?thesis
    proof
      assume "A ∧ B"
      then have "B"..
      with ‹¬B›
      show False..
    qed
  qed
qed
```

Anmerkung: Ist Ihnen bei den Beweisen der De Morgan-Regeln etwas aufgefallen?

deMorgan1 kommt ohne Regeln der klassischen Logik aus, für *deMorgan2* braucht man jedoch Fallunterscheidung. D.h. $\neg (A \wedge B) \longrightarrow \neg A \vee \neg B$ ist in intuitionistischer Logik nicht herleitbar. Dies gilt übrigens auch für die Theoreme $(\neg A \longrightarrow B) \longrightarrow \neg B \longrightarrow A$, $((A \longrightarrow B) \longrightarrow A) \longrightarrow A$, $A \vee \neg A$ und $\neg \neg A \longrightarrow A$