Karlsruher Institut für Technologie

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

Theorembeweiserpraktikum – SS 2013

http://pp.ipd.kit.edu/lehre/SS2013/tba

Blatt 2: Simplifikation

Abgabe: 29. April 2013
Besprechung: 30. April 2013

1 Prädikatenlogik

Es geht wiederum 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: $(\bigwedge x.\ P\ x) \Longrightarrow \forall x.\ P\ x$ *allE*: $[\![\forall x.\ P\ x;\ P\ x \Longrightarrow R]\!] \Longrightarrow R$

exI: $P\ x \Longrightarrow \exists x.\ P\ x$ *exE*: $[\![\exists x.\ P\ x;\ \bigwedge x.\ P\ x \Longrightarrow Q]\!] \Longrightarrow Q$

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 "∀x. P x ⟶ (∃x. P x)"
proof(rule allI)
  fix x
  show "P x ⟶ (∃x. P x)"
  proof
    assume "P x"
    thus "∃x. P x" by (rule exI)
  qed
qed
```

lemma `"(∀x. P x) = (¬ (∃x. ¬ P x))"`
⟨*solution*⟩

lemma `"(¬ (∀x. P x)) = (∃x. ¬ P x)"`
⟨*solution*⟩

lemma `"(∀x. P x ⟶ Q) = ((∃x. P x) ⟶ Q)"`
⟨*solution*⟩

lemma `"(∃x. ∀y. P x y) ⟶ (∀y. ∃x. P x y)"`
⟨*solution*⟩

lemma `"((∀x. P x) ∧ (∀x. Q x)) = (∀x. (P x ∧ Q x))"`
⟨*solution*⟩

lemma *"((∃x. P x) ∨ (∃x. Q x)) = (∃x. (P x ∨ Q x))"*
⟨*solution*⟩

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

lemma *"∃x. P x ⟶ (∀x. P x)"*
⟨*solution*⟩

2 Definitionen und Arbeiten mit Gleicheit

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

definition
nor :: "bool ⇒ bool ⇒ bool" (**infixl** *"↓" 60*)
where *"A ↓ B = ..."*

Nun leiten Sie Lemmas her, die die üblichen boolschen Junktionen nur mit *False* und ↓ darstellen. 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 *¬ (... ∨ ...)* nur Junktionen 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) = ..."*
⟨*solution*⟩

lemma *rewrite_or: "(A ∨ B) = ..."*
⟨*solution*⟩

lemma *rewrite_and: "(A ∧ B) = ..."*
⟨*solution*⟩

lemma *rewrite_imp: "(A ⟶ B) = ..."*
⟨*solution*⟩

lemma *rewrite_True: "True = ..."*
⟨*solution*⟩

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

lemma *"((A ⟶ (A ∨ B)) ∧ ¬ B) = ..."*
⟨*solution*⟩