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

Blatt 2: Simplifikation

Abgabe: 7. Mai 2018, 12:00 Uhr
Besprechung: 8. Mai 2018

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: $(\bigwedge x.\ P\ x) \Longrightarrow \forall x.\ P\ x$ *allE:* $\forall x.\ P\ x \Longrightarrow (P\ x \Longrightarrow R) \Longrightarrow R$

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

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"
    then show "∃x. P x" by (rule exI)
  qed
qed
```

lemma "$(\forall x.\ P\ x) \longleftrightarrow \neg\ (\exists x.\ \neg\ P\ x)$"
⟨*solution*⟩

lemma "$(\neg\ (\forall x.\ P\ x)) \longleftrightarrow (\exists x.\ \neg\ P\ x)$"
⟨*solution*⟩

lemma "$(\forall x.\ P\ x \longrightarrow Q) \longleftrightarrow ((\exists x.\ P\ x) \longrightarrow Q)$"
⟨*solution*⟩

lemma "$(\exists x.\ \forall y.\ P\ x\ y) \longrightarrow (\forall y.\ \exists x.\ P\ x\ y)$"
⟨*solution*⟩

lemma "$(\forall x.\ P\ x) \wedge (\forall x.\ Q\ x) \longleftrightarrow (\forall x.\ (P\ x \wedge Q\ x))$"
⟨*solution*⟩

lemma "$(\exists x.\ P\ x) \vee (\exists x.\ Q\ x) \longleftrightarrow (\exists x.\ (P\ x \vee Q\ x))$"

⟨*solution*⟩

Bei diesem Lemma darf mit Fallunterscheidung (Methode `cases`) 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 Gleichheit

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

definition
`nor :: "bool ⇒ bool ⇒ bool"` `(`**infix** `"↓" 37)` — Eingabe als `\down`
where `"A ↓ B ⟷ ...."`

Nun leiten Sie Lemmas her, die die üblichen boolschen Junktionen nur mit `False` und `op ↓` darstellen. Theoretisch ist `op ↓` 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 `¬ (.... ∨ ....)` 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 schrittweise so um, das er nur mit `op ↓` und `False` gebildet wird. Verwenden Sie dabei das **also** ... **finally** Konstrukt.

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