Karlsruhe Institute of Technology

Lehrstuhl für Programmierparadigmen
Daniel Wasserrab
daniel.wasserrab@kit.edu

Theorembeweiserpraktikum – SS 2010

http://pp.info.uni-karlsruhe.de/lehre/SS2010/tba

Blatt 6: Lesbare Beweise mit Isar Besprechung: 18.05.2010

1 Natürliches Schließen mittels Isar

Wir wollen in dieser Aufgabe einige Aussagen von Blatt 1 nochmals beweisen – diesmal jedoch mittels Verwendung von Isar. Versuchen Sie dabei, das Beweisskript analog Ihren Überlegungen und dabei so übersichtlich und verständlich wie möglich zu halten.

Hinweise:

- Sie dürfen Fallunterscheidungen benutzen, jedoch **nur** in *proof* Statements, also *proof(cases P)*.
- Sie dürfen auch *simp* benutzen, allerdings nur mit Option *only* und vorher bewiesenen Lemmas (also kein *auto*!)
- Aussagen, die durch *from*, *hence*, *thus*, *with* etc. zum Beweis hinzugefügt werden, werden nicht automatisch zu Prämissen. Oftmals muss also vor Anwendung eines *erule* (manchmal auch eines *rule*) erst die Regel mittels eines *apply* – "zusammengefügt" werden.
- *by assumption* kann durch . abgekürzt werden
- Evtl. könnten Sie für Widerspruchsbeweise dieses Lemma brauchen:
  ccontr: *(¬ P ⟹ False) ⟹ P*
- Sie dürfen auch gerne die Reihenfolge der Lemmas verändern, wenn Sie eines zum Beweis eines anderen verwenden wollen

Beispiel:

```
lemma imp_uncurry: "(P ⟶ (Q ⟶ R)) ⟶ P ∧ Q ⟶ R"
proof
  assume pqr:"P ⟶ Q ⟶ R"
  show "P ∧ Q ⟶ R"
  proof
    assume "P ∧ Q"
    hence "P" and "Q" by -(erule conjE,assumption)+
    from pqr `P` have "Q ⟶ R" by(rule mp)
    with `Q` show "R" by -(rule mp)
  qed
qed
```

Und jetzt Sie:

lemma *"A ∧ B ⟶ B ∧ A"*
oops

lemma *"(A ∧ B) ⟶ (A ∨ B)"*
oops

lemma *"((A ∨ B) ∨ C) ⟶ A ∨ (B ∨ C)"*
oops

lemma *"(A ∨ A) = (A ∧ A)"*
oops

lemma *S: "(A ⟶ B ⟶ C) ⟶ (A ⟶ B) ⟶ A ⟶ C"*
oops

lemma *"(A ⟶ B) ⟶ (B ⟶ C) ⟶ A ⟶ C"*
oops

lemma *"¬ ¬ A ⟶ A"*
oops

lemma *"A ⟶ ¬ ¬ A"*
oops

lemma *"(¬ A ⟶ B) ⟶ (¬ B ⟶ A)"*
oops

lemma *"((A ⟶ B) ⟶ A) ⟶ A"*
oops

lemma *"(A ⟶ B) = (¬ A ∨ B)"*
oops

lemma *"(¬ (A ∨ B)) = (¬ A ∧ ¬ B)"*
oops

lemma *"(¬ (A ∧ B)) = (¬ A ∨ ¬ B)"*
oops

2 Quantoren in Isar

Beweisen Sie folgende Aussagen mittels eines strukturierten Beweises in Isar. Sie dürfen zwar automatische Taktiken verwenden, jedoch bitte nicht um das gesamte Lemma zu beweisen, die Beweisidee soll strukturiert aufgeschrieben werden.
Hinweis: wenn Sie eine Aussage und die entsprechend negierte Aussage in Ihren Annahmen haben, können Sie den entsprechenden Ausdruck gleich zu False auswerten lassen mittels der Taktik *contradiction*.

lemma assumes *"∃x. ∀y. P x y"* **shows** *"∀y. ∃x. P x y"*
oops

lemma *"(∀x. P x) = (¬ (∃x. ¬ P x))"*
oops

lemma "(∀x. P x ⟶ Q) = ((∃x. P x) ⟶ Q)"
oops

lemma "∃x. P x ⟶ (∀x. P x)"
oops

3 Rätsel: Der reiche Großvater

Zeigen Sie, dass folgende Aussage gilt:

Wenn jeder arme Mann einen reichen Vater hat,
dann gibt es einen reichen Mann mit einem reichen Großvater

theorem "∀x. ¬ rich x ⟶ rich (father x)
  ⟹ ∃y. rich y ∧ rich(father(father y))"
oops

Hinweise:

- Gibt es überhaupt einen reichen Mann?
- Überlegen Sie sich den Beweis erst auf Papier und versuchen Sie ihn dann möglichst analog in Isar zu formulieren
- Sie werden in jedem Fall Fallunterscheidungen brauchen

4 Cantors Theorem

Sie sollen nun Cantors Theorem beweisen; dieses sagt aus, dass es keine surjektive Funktion von einer Menge auf ihre Potenzmenge geben kann. Formalisiert:

theorem "∃S. S ∉ range (f :: 'a ⇒ 'a set)"

Dabei bezeichnet `range f` die Wertemenge einer Funktion.
Hinweise:

- Der Knackpunkt des Beweises ist das Finden der richtigen Menge `S`. Versuchen Sie es erstmal alleine, erinnern Sie sich (falls bekannt) an das sogenannte *Cantor'sche Diagonalverfahren.* Ansonsten versuchen Sie ihr Glück im Internet, der Name der Übung sollte Hinweis genug sein ;)
- Auch hier sollten Sie sich Ihren Beweis erst auf Papier überlegen und dann möglichst analog in Isar übertragen
- Falls Sie eine Aussage wie `b ∈ range f` haben, lässt sich daraus unmittelbar ein `x` auswählen ("obtainen"), so dass `b = f x` gilt, da die Regel `rangeE:` ⟦`b ∈ range f;` ⋀`x. b = f x` ⟹ `P`⟧ ⟹ `P` als Eliminationsregel in allen Taktiken des automatischen Schließens existiert