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

Theorembeweiserpraktikum – SS 2010

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

Blatt 2: Simplifikation Besprechung: 20.04.2010

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:  (⋀x. P x) ⟹ ∀x. P x
allE:  ⟦∀x. P x; P x ⟹ R⟧ ⟹ R
 exI:  P x ⟹ ∃x. P x
 exE:  ⟦∃x. P x; ⋀x. P x ⟹ Q⟧ ⟹ Q
```

Es dürfen wiederum nur die Methoden *rule*, *erule* und *assumption* verwendet werden. Achten Sie darauf, für *exI* und *allE* nur *rule_tac* bzw. *erule_tac* mit gleichzeitiger Instantiierung der Variablen zu verwenden, ansonsten erhalten sie beliebige Variablen, die den Beweis erschweren können!

Beispiel:

```
lemma "∀x. P x ⟶ (∃x. P x)"
apply (rule allI)
apply (rule impI)
apply (rule_tac x="x" in exI)
apply assumption
done
```

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

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

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

```
lemma "(∃x. ∀y. P x y) ⟶ (∀y. ∃x. P x y)"
oops
```

```
lemma "((∀x. P x) ∧ (∀x. Q x)) = (∀x. (P x ∧ Q x))"
oops
```

```
lemma "((∃x. P x) ∨ (∃x. Q x)) = (∃x. (P x ∨ Q x))"
oops
```

```
lemma "∃x. P x ⟶ (∀x. P x)"
oops
```

2 Prädikatenlogik und Simplifikation

In klassischer Aussagenlogik können die Operatoren $=$, $\vee$, $\neg$ durch $\longrightarrow$, $\wedge$, *False* ersetzt werden. Definieren und beweisen Sie entsprechende Simplifikationslemmas. Benutzen Sie dafür wieder nur die Methoden *rule*, *erule* und *assumption* und die bisher bekannten Regeln, zusätzlich noch folgende:

FalseE: $\;$ *False* $\Longrightarrow$ *P*
TrueI: $\;$ *True*

lemma *rewrite_not:"(¬ A) = ..."*
oops

lemma *rewrite_eq:"(A = B) = ..."*
oops

lemma *rewrite_or:"(A ∨ B) = ..."*
oops

Versuchen Sie nun, folgende Ausdrücke soweit wie möglich umzuschreiben. Benutzen Sie dazu die Methode *simp* mit der Option *only*. Sie können danach versuchen, das Resultat zu beweisen (natürlich nur mit den Regeln für $\longrightarrow$ bzw. $\wedge$) oder den Beweis einfach mittels *oops* abbrechen.

lemma *"(A ∧ True) = A"*
oops

lemma *"(A ∨ B) = (¬ A ⟶ B)"*
oops

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

3 Simplifikation am Beispiel xor

In dieser Aufgabe soll ein neuer Operator *xor* bzw. $\oplus$ wie üblich definiert werden. Anschließend werden ein paar Termersetzungsregeln formuliert und bewiesen und schließlich an einem größeren Beispiel das Verhalten des Simplifiers simuliert.

Definieren Sie zunächst analog zum *nand* Beispiel aus den Folien den Operator *xor*:

definition
  xor :: "bool ⇒ bool ⇒ bool" (**infixl** *"⊕" 60)*
  where *"A ⊕ B ≡ ...."*

Jetzt beweisen wir ein paar Simplifikationsregeln, indem wir als erstes die Definition von *xor* auffalten mittels `apply(simp only:xor_def)` und das Resultat dann mit den aus den bisherigen Übungen bekannten Methoden beweisen.

lemma *xor_True:"True ⊕ A = (¬ A)"*
— Auffalten der Definition, auch in den folgenden Lemmas jeweils erster Schritt
apply *(simp only:xor_def)*
— Und jetzt der Beweis
oops

lemma *xor_False:"False ⊕ A = A"*
oops

— Sie benötigen evtl. Fallunterscheidung nach A, um dies zu beweisen
lemma *xor_asym_aux:"A ⊕ (¬ A)"*
oops

lemma *xor_not_sym_aux:"¬ A ⊕ A"*
oops

Die letzten beiden Lemmas können wir nicht als Termersetzungsregeln verwenden, da sie nicht die benötigte Gleichungsform haben. Jedoch kann man jetzt die benötigten Simplifikationslemmas formulieren und jene verwenden, um sie zu beweisen.

— einfach Beweis 'entkommentieren' und oops entfernen
lemma *xor_asym:"A ⊕ (¬ A) = True"*

oops

lemma *xor_not_sym:"A ⊕ A = False"*

oops

Und zum Schluß noch eine etwas komplizierter zu zeigende Regel:

— Auch hier können Fallunterscheidungen nötig sein
lemma *xor_simp:"A ⊕ (A ⊕ B) = B"*
oops

Beim folgenden Lemma sollen nun Sie den Simplifier direkt anleiten. Sie dürfen dafür alle obigen Termersetzungsregeln verwenden, aber jedoch nur eine pro Schritt, also:
`apply(simp only:` $Regel_x$`)`
`apply(simp only:` $Regel_y$`)`
...

lemma *"(B ⊕ ((((A ⊕ (A ⊕ (¬ B))) ⊕ (¬ B)) ⊕ B) ⊕ A)) ⊕*
(((¬ B) ⊕ ((A ⊕ (¬ A)) ⊕ B)) ⊕ A) = False"
oops