Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph. D.
Dr. Stefan Berghofer

SoS 2005
31. 5. 2005
Aufgabenblatt 5

Vorlesung Perlen der Informatik II

Natürliches Schließen

Auf diesem Aufgabenblatt geht es um den Kalkül des natürlichen Schließens, mit dessen Hilfe einige Lemmas der Aussagen- und Prädikatenlogik bewiesen werden sollen.

Für die Beweise gelten die folgenden Spielregeln:

- Es dürfen nur diese Regeln verwendet werden:

  notI: `(A ⟹ False) ⟹ ¬ A`
  notE: `⟦¬ A; A⟧ ⟹ B`
  conjI: `⟦A; B⟧ ⟹ A ∧ B`
  conjE: `⟦A ∧ B; ⟦A; B⟧ ⟹ C⟧ ⟹ C`
  disjI1: `A ⟹ A ∨ B`
  disjI2: `A ⟹ B ∨ A`
  disjE: `⟦A ∨ B; A ⟹ C; B ⟹ C⟧ ⟹ C`
  impI: `(A ⟹ B) ⟹ A ⟶ B`
  impE: `⟦A ⟶ B; A; B ⟹ C⟧ ⟹ C`
  mp: `⟦A ⟶ B; A⟧ ⟹ B`
  iffI: `⟦A ⟹ B; B ⟹ A⟧ ⟹ A = B`
  iffE: `⟦A = B; ⟦A ⟶ B; B ⟶ A⟧ ⟹ C⟧ ⟹ C`
  exI: `P x ⟹ ∃x. P x`
  exE: `⟦∃x. P x; ⋀x. P x ⟹ Q⟧ ⟹ Q`
  allI: `(⋀x. P x) ⟹ ∀x. P x`
  allE: `⟦∀x. P x; P x ⟹ R⟧ ⟹ R`

- Es dürfen nur die Methoden *(rule r)*, *(erule r)* und *assumption* verwendet werden, wobei r eine der angegebenen Regeln ist. Zur Fallunterscheidung können sie die Methode *(case_tac A)*, wobei A ein boolscher Ausdruck (z.B. eine Variable) ist. Ausserdem Sie dürfen die Kommandos *prefer* n und *back* verwenden.

Aufgabe 1 (Aussagenlogik)

lemma `"((A ∨ B) ∨ C) ⟶ A ∨ (B ∨ C)"` ⟨*proof*⟩
lemma `"(A ∨ A) = (A ∧ A)"` ⟨*proof*⟩
lemma `S: "(A ⟶ B ⟶ C) ⟶ (A ⟶ B) ⟶ A ⟶ C"` ⟨*proof*⟩
lemma `"((A ⟶ B) ⟶ A) ⟶ A"` ⟨*proof*⟩
lemma `"(A ⟶ (B ∧ C)) ⟶ (B ⟶ ¬ C) ⟶ ¬ A"` ⟨*proof*⟩
lemma `"(D ⟶ A) ⟶ (A ⟶ (B ∧ C)) ⟶ (B ⟶ ¬ C) ⟶ ¬ D"` ⟨*proof*⟩
lemma `"(A ⟶ ¬ B) = (B ⟶ ¬ A)"` ⟨*proof*⟩
lemma `"((A ∧ ¬ B) ∨ (B ∧ ¬ A)) = (A = (¬ B))"` ⟨*proof*⟩

Aufgabe 2 (Prädikatenlogik)

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

Aufgabe 3 (Löschen in binären Bäumen)

Binäre Bäume, die Daten in ihren inneren Knoten abspeichern (in diesem Fall sind diese Daten vom Typ `nat`), können wie folgt in Isabelle/HOL repräsentiert werden:

datatype `tree = Tip | Nd tree nat tree`

Definieren Sie eine Funktion

consts `set_of :: "tree ⇒ nat set"`

die einen solchen binären Baum als Argument nimmt und die Menge der Daten zurückliefert, die im Baum gespeichert sind.

Definieren Sie weiterhin eine Funktion

consts `sorted :: "tree ⇒ bool"`

die angibt, ob die Daten in einem binären Baum von links nach rechts strict monoton aufsteigend sortiert sind.

Definieren Sie schliesslich eine Funktion, die ein Datum (eine gegebene natürliche Zahl) aus einem *sortierten* Baum herauslöscht, und wieder einen sortierten Baum als Ergebnis zurückliefert:

consts
`remove:: "nat ⇒ tree ⇒ tree"`

TIP: Führen Sie Hilfsfunktionen ein, die bei der Definition von `remove` nützlich sind. Diese Funktionen müssen nicht total (definiert) sein!

Überlegen Sie sich, was es für Ihre `remove`-Funktion bedeutet, korrekt zu sein. Beweisen Sie dann, dass Ihre Funktion korrekt ist.

▷ **Abgabe: 9. 6. 2005**