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 7: Induktion in Isar Besprechung: 25.05.2010

1 Rotation mal ungewöhnlich

Wir definieren eine Funktion auf Listen, welche das erste Element an die letzte Stelle schiebt:

```
fun rot :: "'a list ⇒ 'a list"
where "rot (x#y#zs) = y # rot(x#zs)"
  | "rot xs = xs"
```

Die von Isabelle automatisch erstellte Rekursionsregel lautet `rot.induct:`
`⟦⋀x y zs. P (x # zs) ⟹ P (x # y # zs); P []; ⋀v. P [v]⟧ ⟹ P a0`
Damit beweisen Sie bitte folgende Aussagen mittels eines *verständlichen* Isar-Beweises:

```
lemma "length (rot xs) = length xs"
oops

lemma "xs ≠ [] ⟹ rot xs = tl xs @ [hd xs]"
oops
```

2 Reflexiv-transitive Hülle

Wir definieren die reflexiv-transitive Hülle einer binären Prädikats `r` mittels eines induktiven Prädikats:

```
inductive rtc :: "('a ⇒ 'a ⇒ bool) ⇒ 'a ⇒ 'a ⇒ bool" ("(_*)" [1000] 1000)
for r::"'a ⇒ 'a ⇒ bool"
where refl: "r* x x"
  | step: "⟦r x y; r* y z⟧ ⟹ r* x z"
```

Anstatt `rtc r` darf man also `r*` schreiben. Auch hier generiert Isabelle automatisch eine Induktionsregel `rtc.induct: ⟦r* x1 x2; ⋀x. P x x; ⋀x y z. ⟦r x y; r* y z; P y z⟧ ⟹ P x z⟧ ⟹ P x1 x2`. Zeigen Sie jetzt, dass `r*` tatsächlich transitiv ist:

```
lemma "⟦r* x y; r* y z⟧ ⟹ r* x z"
oops
```

Außerdem beweisen Sie noch, dass `r*` idempotent (die reflexiv-transitive Hülle von `r*` gleich `r*`) ist. Dazu brauchen Sie folgende Aussage: `ext: (⋀x. ?f x = ?g x) ⟹ ?f = ?g`

```
lemma rtc_idemp: "(r*)* = r*"
proof(rule ext)+
oops
```

Alle Beweise sollen natürlich mittels Isar erstellt werden.

3 Kontextfreie Grammatiken für Klammerausdrücke

Im folgenden soll eine Grammatik für Klammerausdrücke (als Listen) formalisiert werden. Solche Grammatiken haben üblicherweise folgende Darstellung:

$$S \quad \rightarrow \quad \varepsilon \quad | \quad '(' \; S \; ')' \quad | \quad S \; S$$

Sie sollen nun diese Grammatik als induktives Prädikat definieren. Dabei soll der folgende Datentyp die Klammern beschreiben, `A` eine öffnende, `B` eine schließende Klammer (die normalen Klammern sind in Isabelle funktional überladen):

datatype `brack = A | B`

Man kann auch noch auf eine andere Weise prüfen, ob Klammerausdrücke korrekt sind: durch Abzählen der öffnenden und schließenden Klammern. Für jede öffnende Klammer inkrementiert man einen Zähler, für jede schließende dekrementiert man ihn. Wenn der Zähler 0 ist, darf er nicht weiter dekrementiert werden!

Definieren Sie ein Prädikat zur Prüfung von Klammerausdrücken, basierend auf dieser Methode.

Hinweise:

- verwenden sie statt `n + 1` `Suc n`
- wenn eine beliebige natürliche Zahl von 0 abgezogen wird, ist das Resultat wieder 0
- wie muss der Zähler beschaffen sein, wenn man auf eine schließende Klammer trifft?

Und nun beweisen Sie, dass jeder Parameter, der das oben definierte induktive Prädikat erfüllt auch dieses Prädikat erfüllt.