

Praktikum Spezifikation und Verifikation

Funktionen auf Listen (Lösungsvorschlag)

`theory 11 = Main:`

Definieren Sie den All- und den Existenz-Quantor für Listen. Die Funktion `alls P xs` soll genau dann wahr sein, wenn `P x` für jedes Element `x` der Liste `xs` wahr ist; analog soll `exs P xs` wahr sein, wenn `P` für mindestens ein Element wahr ist.

`consts`

```
alls :: "('a ⇒ bool) ⇒ 'a list ⇒ bool"
exs  :: "('a ⇒ bool) ⇒ 'a list ⇒ bool"
```

`primrec`

```
"alls P [] = True"
"alls P (x#xs) = (if ¬P x then False else alls P xs)"
```

`primrec`

```
"exs P [] = False"
"exs P (x#xs) = (if P x then True else exs P xs)"
```

Zeigen oder widerlegen Sie (mit Gegenbeispiel) folgende Lemmas. Sie benötigen unter Umständen zusätzliche Lemmas, damit der Beweis funktioniert. Benutzen Sie das `[simp]`-Attribut nur, wenn die Gleichung, die Sie zeigen, eine wirkliche Vereinfachung ist, oder Sie die Aussage als Lemma für einen späteren Beweis brauchen.

```
lemma [simp]: "alls (λx. P x ∧ Q x) xs = (alls P xs ∧ alls Q xs)"
  apply (induct_tac xs)
  apply auto
  done
```

```
lemma [simp]: "alls P (a@b) = (alls P a ∧ alls P b)"
  apply (induct_tac a)
  apply auto
  done
```

```

lemma "alls P (rev xs) = alls P xs"
  apply (induct_tac xs)
  apply auto
  done

```

```

lemma "exs ( $\lambda x. P x \wedge Q x$ ) xs = (exs P xs  $\wedge$  exs Q xs)"
  apply (induct_tac xs)
  apply auto
  :

```

```

lemma "a $\neq$ b  $\implies$ 
  exs ( $\lambda x. x=a \wedge x=b$ ) [a,b]  $\neq$  (exs ( $\lambda x. x=a$ ) [a,b]  $\wedge$  exs ( $\lambda x. x=b$ ) [a,b])"
  apply auto
  done

```

```

lemma "exs P (map f xs) = exs (P o f) xs"
  apply (induct_tac xs)
  apply auto
  done

```

```

lemma [simp]: "exs P (a@b) = (exs P a  $\vee$  exs P b)"
  apply (induct_tac a)
  apply auto
  done

```

```

lemma "exs P (rev xs) = exs P xs"
  apply (induct_tac xs)
  apply auto
  done

```

Finden Sie ein Z, so dass folgende Gleichung gilt:

```

lemma "exs ( $\lambda x. P x \vee Q x$ ) xs = exs P xs  $\vee$  exs Q xs"
  apply (induct_tac xs)
  apply auto
  done

```

Drücken Sie den Existenzquantor durch den Allquantor aus (auf der rechten Seite soll kein *exs* mehr vorkommen).

```

lemma "exs P xs = ( $\neg$ alls ( $\lambda x. \neg P x$ ) xs)"
  apply (induct_tac xs)
  apply auto
  done

```

Definieren Sie eine Funktion *is_in x xs*, die testet ob *x* in der Liste *xs* vorkommt.

```
consts
  is_in :: "'a ⇒ 'a list ⇒ bool"
```

```
primrec
```

```
"is_in y [] = False"
"is_in y (x#xs) = (if x = y then True else is_in y xs)"
```

Drücken Sie dann die Funktion `is_in` mit Hilfe des Existenzquantors auf Listen aus, und beweisen Sie, dass ihre Gleichung stimmt.

```
lemma "is_in a xs = exs (λx. a = x) xs"
  apply (induct_tac xs)
  apply auto
  done
```

Schreiben Sie eine Funktion `nodups xs`, die genau dann wahr ist, wenn jedes Element von `xs` nur einmal in der Liste vorkommt. Definieren Sie zusätzlich eine Funktion `deldups xs`, die Duplikate aus Listen entfernt.

Zeigen oder widerlegen Sie dann (mit Gegenbeispiel) folgende Lemmas.

```
consts
```

```
nodups :: "'a list ⇒ bool"
deldups :: "'a list ⇒ 'a list"
```

```
primrec
```

```
"nodups [] = True"
"nodups (x#xs) = (if is_in x xs then False else nodups xs)"
```

```
primrec
```

```
"deldups [] = []"
"deldups (x#xs) = (if is_in x xs then deldups xs else x#deldups xs)"
```

```
lemma "length (deldups xs) ≤ length xs"
  apply (induct_tac xs)
  apply auto
  done
```

```
lemma [simp]: "is_in a (deldups xs) = is_in a xs"
  apply (induct_tac xs)
  apply auto
  done
```

```
lemma "nodups (deldups xs)"
  apply (induct_tac xs)
  apply auto
  done
```

```
lemma "deldups (rev xs) = rev (deldups xs)"
```

```
apply (induct_tac xs)
apply auto
:
```

lemma

```
"a ≠ c ⇒ deldups (rev [a,a,c,a]) ≠ rev (deldups [a,a,c,a])"
apply auto
done
```