Univerzitet u Beogradu
Matematički fakultet
Katedra za računarstvo i informatiku

Šifra predmeta: R265
03.05.2023.

Uvod u interaktivno dokazivanje teorema

Vežbe 10

Zadatak 1 *Tip: list.*

Diskutovati o sledećim termovima i vrednostima.

term []
term *1* # *2* # []
term (*1*::*nat*) # *2* # []
term [*1*, *2*]
term [*1*::*nat*, *2*]

value [*1*..*5*]
value [*1*..<*5*]

term *sum-list*
value *sum-list* [*1*..<*5*]

term *map*
term λ *x*. *f x*
value *map* (λ *x*. *x*^*2*) [*1*..<*5*]
value *sum-list* (*map* (λ *x*. *x*^*2*) [*1*..<*5*])

value $\sum$ *x* $\leftarrow$ [*1*..<*5*]. *x*^*2*

Zadatak 2 *Sumiranje nizova preko listi.*

Pokazati da važi: $1 + 2^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}$.

primrec *zbir-kvadrata* :: *nat* $\Rightarrow$ *nat* **where**
  zbir-kvadrata 0 = *0*
| *zbir-kvadrata* (*Suc n*) = *zbir-kvadrata n* + (*Suc n*) ^ *2*

Definisati funkciju *zbir-kvadrata'* :: *nat* $\Rightarrow$ *nat* preko definicije, koja računa levu stranu jednakosti pomoću liste i funkcijama nad listama.

definition *zbir-kvadrata'* :: *nat* $\Rightarrow$ *nat* **where**
  zbir-kvadrata' n = *sum-list* (*map* (λ *x*. *x*^*2*) [*1*..<*Suc n*])

Pokazati da su ove dve funkcije ekvivalentne.

lemma *zbir-kvadrata n* = *zbir-kvadrata' n*
  by (*induction n*) (*auto simp add*: *zbir-kvadrata'-def*)

Pokazati automatski da je *zbir-kvadrata n* = *n* ∗ (*n* + *1*) ∗ (*2* ∗ *n* + *1*) *div 6*.
Savet: Razmotriti leme koje se koriste u Isar verziji dokaza i dodati ih u *simp*.

lemma *zbir-kvadrata n* = *n* ∗ (*n* + *1*) ∗ (*2* ∗ *n* + *1*) *div 6*
  by (*induction n*) (*auto simp add*: *algebra-simps power2-eq-square*)

Zadatak 3 *Algebarski tip podataka: lista.*

Definisati polimorfan algebarski tip podataka *'a lista* koji predstavlja listu elemenata polimorfong tipa *'a*.

datatype *'a lista = Prazna*
| Dodaj 'a 'a lista

term *Dodaj (1::nat) (Dodaj 2 (Dodaj 3 Prazna))*

Definisati funkcije *duzina' :: 'a lista ⇒ nat*, *nadovezi' :: 'a lista ⇒ 'a lista ⇒ 'a lista*, *obrni' :: 'a lista ⇒ 'a lista* primitivnom rekurzijom koje računaju dužinu liste, nadoveziju i obrću liste tipa *'a lista*.

primrec *duzina' :: 'a lista ⇒ nat* **where**
duzina' Prazna = 0
| duzina' (Dodaj - xs) = 1 + duzina' xs

primrec *nadovezi' :: 'a lista ⇒ 'a lista ⇒ 'a lista* **where**
nadovezi' Prazna ys = ys
| nadovezi' (Dodaj x xs) ys = Dodaj x (nadovezi' xs ys)

primrec *obrni' :: 'a lista ⇒ 'a lista* **where**
obrni' Prazna = Prazna
| obrni' (Dodaj x xs) = nadovezi' (obrni' xs) (Dodaj x Prazna)

Definisati funkciju *duzina :: 'a list ⇒ nat* primitivnom rekurzijom koja računa dužinu liste tipa *'a list*. Ta pokazati da su *duzina* i *length* ekvivalentne funkcije.

primrec *duzina :: 'a list ⇒ nat* **where**
duzina [] = 0
| duzina (x # xs) = 1 + duzina xs

lemma *duzina-length*:
shows *duzina xs = length xs*
by *(induction xs) auto*

Definisati funkciju *prebroj :: ('a::equal) ⇒ 'a list ⇒ nat* primitivnom rekurzijom koja računa koliko se puta javlja element tipa *'a::equal* u listi tipa *('a::equal) list*. Ta pokazati da je *prebroj a xs ≤ length xs*.

primrec *prebroj :: ('a::equal) ⇒ 'a list ⇒ nat* **where**
prebroj a []= 0
| prebroj a (x # xs)= (if a = x then 1 + prebroj a xs else prebroj a xs)

lemma *prebroj a xs ≤ length xs*
by *(induction xs) auto*

term *count-list*

Definisati funkicju *sadrzi :: ('a::equal) ⇒ 'a list ⇒ bool* primitivnom rekurzijom koja ispituje da li se element tipa *'a::equal* javlja u listi tipa *('a::equal) list*. Ta pokazati da je *sadrzi a xs = a ∈ set xs*

primrec *sadrzi :: ('a::equal) ⇒ 'a list ⇒ bool* **where**
sadrzi a [] ⟷ False

| *sadrzi a* (*x* # *xs*) $\longleftrightarrow$ *a* = *x* $\vee$ *sadrzi a xs*

lemma *sadrzi a xs* $\longleftrightarrow$ *a* $\in$ *set xs*
  by (*induction xs*) *auto*

Definisati funkciju *skup* :: *'a list* $\Rightarrow$ *'a set* primitivnom rekurzijom koja vraća skup tipa *'a set* koji je sačinjen od elemenata liste tipa *'a list*. Ta pokazati da je *skup xs* = *set xs*.

primrec *skup* :: *'a list* $\Rightarrow$ *'a set* **where**
  skup [] = {}
| *skup* (*x* # *xs*) = {*x*} $\cup$ *skup xs*

Definisati funkciju *nadovezi* :: *'a list* $\Rightarrow$ *'a list* $\Rightarrow$ *'a list* primitivnom rekurzijom koja nadovezuje jednu listu na drugu tipa *'a list*. Ta pokazati da je ekvivalentna ugrađenoj funkciji *append* ili infiksom operatoru @.

primrec *nadovezi* :: *'a list* $\Rightarrow$ *'a list* $\Rightarrow$ *'a list* **where**
  nadovezi [] *ys* = *ys*
| *nadovezi* (*x* # *xs*) *ys* = *x* # *nadovezi xs ys*

lemma *nadovezi-append*:
  shows *nadovezi xs ys* = *xs* @ *ys*
  by (*induction xs*) *auto*

Formulisati i pokazati da je dužina dve nedovezane liste, zbir dužina pojedinačnih listi.
Orediti i dokazati osobine za funkcije *skup* i *nadovezi*, kao i za *sadrzi* i *nadovezi*.

lemma *duzina-nadovezi*:
  shows *duzina* (*nadovezi xs ys*) = *duzina xs* + *duzina ys*
  by (*induction xs*) *auto*

lemma *skup-nadovezi*:
  shows *skup* (*nadovezi xs ys*) = *skup xs* $\cup$ *skup ys*
  by (*induction xs*) *auto*

lemma *sadrzi-nadovezi*:
  shows *sadrzi a* (*nadovezi xs ys*) = *sadrzi a xs* $\vee$ *sadrzi a ys*
  by (*induction xs*) *auto*

Definisati funkicju *obrni* :: *'a list* $\Rightarrow$ *'a list* primitivnom rekurzijom koja obrće listu tipa *'a list*. Ta pokazati da funkcija je *obrni* ekvivalentna funkciji *rev*. Nakon toga pokazati da je dvostruko obrnuta lista ekvivalentna početnoj listi.
Napomena: Pri definisanju funkcije *obrni* nije dozvoljeno koristiti operator nadovezivanje @.
Savet: Potrebno je definisati pomoćne leme.

primrec *obrni* :: *'a list* $\Rightarrow$ *'a list* **where**
  obrni [] = []
| *obrni* (*x* # *xs*) = *nadovezi* (*obrni xs*) [*x*]

lemma *obrni-rev*:
  shows *obrni xs* = *rev xs*
  by (*induction xs*) (*auto simp add*: *nadovezi-append*)

lemma *nadovezi-asoc*:
  shows *nadovezi* (*nadovezi xs ys*) *zs* = *nadovezi xs* (*nadovezi ys zs*)

```
  by (induction xs) auto

lemma nadovezi-Nil-desno [simp]:
  shows nadovezi xs [] = xs
  by (induction xs) auto

lemma obrni-nadovezi [simp]:
  shows obrni (nadovezi xs ys) = nadovezi (obrni ys) (obrni xs)
  by (induction xs) (auto simp add: nadovezi-asoc)

lemma obrni-obrni-id: obrni (obrni xs) = xs
  by (induction xs) auto
```

Definisati funkciju *snoc* :: *'a* ⇒ *'a list* ⇒ *'a list* koja dodaje element na kraj liste, i funkciju *rev-snoc* :: *'a list* ⇒ *'a list* koja uz pomoć funkcije *snoc* obrće elemente liste. Da li *rev-snoc* popravlja složenost obrtanja liste?

```
primrec snoc :: 'a ⇒ 'a list ⇒ 'a list where
  snoc a [] = [a]
| snoc a (x # xs) = x # snoc a xs

primrec rev-snoc :: 'a list ⇒ 'a list where
  rev-snoc [] = []
| rev-snoc (x # xs) = snoc x (rev-snoc xs)
```

Definisati funkciju *itrev* koja obrće listu iterativno.
Savet: Koristiti pomoćnu listu.

```
primrec itrev :: 'a list ⇒ 'a list ⇒ 'a list where
  itrev [] ys = ys
| itrev (x # xs) ys = itrev xs (x # ys)
```

Pokazati da je funkcija *itrev* ekvivalentna ugrađenoj funkciji *rev*, kada je inicijalna pomoćna lista prazna.

```
lemma itrev-rev-append:
  shows itrev xs ys = rev xs @ ys
  by (induction xs arbitrary: ys) auto

lemma itrev-rev:
  shows itrev xs [] = rev xs
  by (induction xs) (auto simp add: itrev-rev-append)
```

Pomoću funkcije *fold* opisati obrtanje liste. Pokazati ekvivalentnost funkciji *itrev* sa obrtanjem liste preko *fold*-a.

```
term fold

lemma fold-Cons-append:
  shows fold (#) xs ys @ zs = fold (#) xs (ys @ zs)
  by (induction xs arbitrary: ys zs) auto

lemma itrev-fold-Cons:
  shows itrev xs ys = fold (#) xs ys
  by (induction xs arbitrary: ys) (auto simp add: itrev-rev-append fold-Cons-append)
```