Univerzitet u Beogradu
Matematički fakultet
Katedra za računarstvo i informatiku

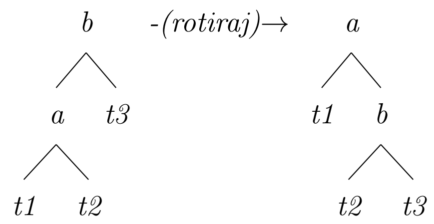
Šifra predmeta: R265
27.05.2024.

Uvod u interaktivno dokazivanje teorema

Vežbe 12

Zadatak 1 *Desni linearni lanac*

Data je funkcija *rotiraj* koja rotira drvo na sledeći način:



$\textbf{fun}\ \mathit{rotiraj} :: {'a\ tree} \Rightarrow {'a\ tree}\ \textbf{where}$
$\quad \mathit{rotiraj}\ \langle\langle t1,\ a,\ t2\rangle,\ b,\ t3\rangle = \langle t1,\ a,\ \langle t2,\ b,\ t3\rangle\rangle$
$|\ \mathit{rotiraj}\ x = x$

thm *rotiraj.simps*

1. *(2p)*

   Definisati funkciju $bps :: ('a::linorder)\ tree \Rightarrow bool$ koja proverava da li je stablo binarno pretraživačko. Za relaciju poretka koristiti $<$, a za konstruisanje skupa elemenata stabla koristiti *set-tree*.

   Sledeća dva izraza moraju da se evaluiraju u *True*:

   $\textbf{value}\ bps\ \langle\langle\langle\langle\rangle,\ 1::nat,\ \langle\rangle\rangle,\ 2,\ \langle\rangle\rangle,\ 5,\ \langle\langle\rangle,\ 6,\ \langle\rangle\rangle\rangle$
   $\textbf{value}\ \neg\ bps\ \langle\langle\langle\langle\rangle,\ 3::nat,\ \langle\rangle\rangle,\ 2,\ \langle\rangle\rangle,\ 5,\ \langle\langle\rangle,\ 6,\ \langle\rangle\rangle\rangle$

2. *(2p)*

   Pokazati da je funkcija *rotiraj* korektna, tj. da funkcija *rotiraj* ne ruši svostvo binarnog pretraživačkog stabla i da skup čvorova ostaje nepromenjen nakon primene funkcije *rotiraj*.

   $\textbf{lemma}\ bps\ t \Longrightarrow bps\ (rotiraj\ t)$

   $\textbf{lemma}\ \textit{set-tree}\ (rotiraj\ t) = \textit{set-tree}\ t$

3. *(3p)*

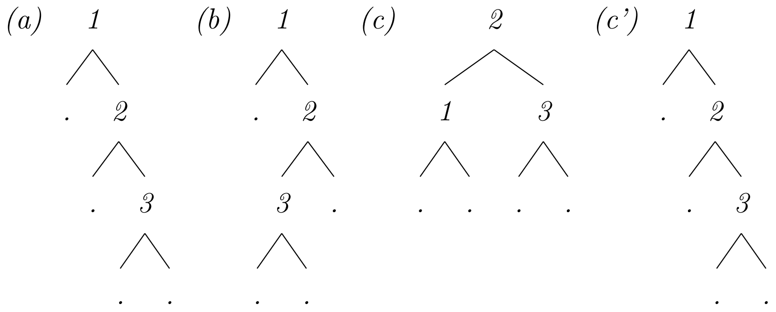
   Binarno stablo ima svojstvo *desnog linearnog lanaca* kada svaki unutrašnji čvor ima samo desnog potomka, tj. binarno stablo koje predstavlja listu duž desne strane. Definisati funkciju $dll :: {'a\ tree} \Rightarrow bool$ koja proverava da li je binarno stablo desni linearni lanac. *Savet*: Izbegavati *if−then−else* izraze, moguće je koristiti *pattern matching*.

   Sledeći izrazi moraju se evaluirati u *True* (pogledati i odgovarajuće slike).

value $\mathit{dll}\ \langle\langle\rangle,\ \mathit{1}{::}\mathit{nat},\ \langle\langle\rangle,\ \mathit{2},\ \langle\langle\rangle,\ \mathit{3},\ \langle\rangle\rangle\rangle\rangle$ — (slika (a))

value $\neg\ \mathit{dll}\ \langle\langle\rangle,\ \mathit{1}{::}\mathit{nat},\ \langle\langle\langle\rangle,\ \mathit{3},\ \langle\rangle\rangle,\ \mathit{2},\ \langle\rangle\rangle\rangle$ — (slika (b))

value $\mathit{dll}\ (\mathit{rotiraj}\ \langle\langle\langle\rangle,\ \mathit{1}{::}\mathit{nat},\ \langle\rangle\rangle,\ \mathit{2},\ \langle\langle\rangle,\ \mathit{3},\ \langle\rangle\rangle\rangle)$ — (slika (c) — pre rotiranja; slika (c') – nakon rotiranja).



4. *(2p)*

Definisati funkciju $\mathit{rotiraj1} :: {'a}\ \mathit{tree} \Rightarrow {'a}\ \mathit{tree}$ koja obilazi stablo i vrši prvu moguću rotaciju.

Sledeći izraz mora da se evaluira u *True* (nakon 3 primene *rotiraj1* nema promena u stablu):

value $(\mathit{rotiraj1}\ \hat{}\hat{}\ \mathit{3})\ \langle\langle\langle\rangle,\ \mathit{3}{::}\mathit{nat},\ \langle\langle\rangle,\ \mathit{5}{::}\mathit{nat},\ \langle\langle\rangle,\ \mathit{6}{::}\mathit{nat},\ \langle\rangle\rangle\rangle\rangle,\ \mathit{1}{::}\mathit{nat},\ \langle\langle\rangle,\ \mathit{2}{::}\mathit{nat},\ \langle\rangle\rangle\rangle$
$\quad = (\mathit{rotiraj1}\ \hat{}\hat{}\ \mathit{10})\ \langle\langle\langle\rangle,\ \mathit{3}{::}\mathit{nat},\ \langle\langle\rangle,\ \mathit{5}{::}\mathit{nat},\ \langle\langle\rangle,\ \mathit{6}{::}\mathit{nat},\ \langle\rangle\rangle\rangle\rangle,\ \mathit{1}{::}\mathit{nat},\ \langle\langle\rangle,\ \mathit{2}{::}\mathit{nat},\ \langle\rangle\rangle\rangle$

5. *(2p)*

Želimo da dokažemo činjenicu da je najviše potrebno *size t* rotacija da bi binarno stablo postalo desni linearni lanac.

Kako bi ovo pokazali moramo definisati meru zaustavljanja, tj. funkciju koja linearno opada u odnosu na broj primena funkcije *rotiraj1* i dostiže *0* kada binarno stablo postane desni linearni lanac. Jedna takva funkcija može biti definisana kao:

fun $\mathit{mera} :: {'a}\ \mathit{tree} \Rightarrow \mathit{nat}$ **where**

$\ \ \mathit{mera}\ \langle\rangle = \mathit{0}$

$|\ \mathit{mera}\ \langle l,\ a,\ r\rangle = \mathit{size}\ l + \mathit{mera}\ r$

Pokazati da stablo t koje je desni linearni lanac uzima vrednost mere zaustavljanja 0.

lemma *mera-0*: $\mathit{dll}\ t \longleftrightarrow \mathit{mera}\ t = \mathit{0}$

6. *(2p)*

Pokazati da se mera smanjuje za 1 nakon primene funkcije *rotiraj1*.

lemma *mera-rotiraj-1*: $\mathit{mera}\ (\mathit{rotiraj1}\ t) = \mathit{mera}\ t - \mathit{1}$

Pokazati da se mera smanjuje za n nakon n primena funkcija *rotiraj1*.

lemma *mera-rotiraj-n*: $\mathit{mera}\ ((\mathit{rotiraj1}\ \hat{}\hat{}\ n)\ t) = \mathit{mera}\ t - n$

7. *(2p)*

   Konačno pokazati da:

   theorem *dll-rotiraj*: $\exists\, n \le$ *size t. dll* ((*rotiraj1* ^^ *n*) *t*)

Rešenje 1

primrec *bps* :: ('*a*::*linorder*) *tree* ⇒ *bool* **where**
  bps ⟨⟩ ⟷ *True*
| *bps* ⟨*l*, *a*, *r*⟩ ⟷ (∀ *x*∈*set-tree l*. *x* < *a*) ∧ (∀ *x*∈*set-tree r*. *a* < *x*) ∧ *bps l* ∧ *bps r*

lemma *bps t* ⟹ *bps* (*rotiraj t*)
  by (*induction t rule*: *rotiraj.induct*) *auto*

lemma *set-tree* (*rotiraj t*) = *set-tree t*
  by (*induction t rule*: *rotiraj.induct*) *auto*

fun *dll* :: '*a tree* ⇒ *bool* **where**
  dll ⟨⟩ = *True*
| *dll* ⟨⟨⟩, *x*, *r*⟩ = *dll r*
| *dll* - = *False*

value *dll* ⟨⟨⟩, *1*::*nat*, ⟨⟨⟩, *2*, ⟨⟨⟩, *3*, ⟨⟩⟩⟩⟩ — (slika (a))
value ¬ *dll* ⟨⟨⟩, *1*::*nat*, ⟨⟨⟨⟩, *3*, ⟨⟩⟩, *2*, ⟨⟩⟩⟩ — (slika (b))
value *dll* (*rotiraj* ⟨⟨⟨⟩, *1*::*nat*, ⟨⟩⟩, *2*, ⟨⟨⟩, *3*, ⟨⟩⟩⟩) — (slika (c) — pre rotiranja; slika (c') – nakon rotiranja).

fun *rotiraj1* :: '*a tree* ⇒ '*a tree* **where**
  rotiraj1 ⟨⟩ = ⟨⟩
| *rotiraj1* ⟨⟨⟩, *a*, *r*⟩ = ⟨⟨⟩, *a*, *rotiraj1 r*⟩
| *rotiraj1* ⟨*l*, *a*, *r*⟩ = *rotiraj* ⟨*l*, *a*, *r*⟩

lemma *mera-0*: *dll t* ⟷ *mera t* = *0*
  by (*induction t rule*: *dll.induct*) *auto*

lemma *mera-rotiraj-1*[*simp*]: *mera* (*rotiraj1 t*) = *mera t* − *1*
  by (*induction t rule*: *rotiraj1.induct*) *auto*

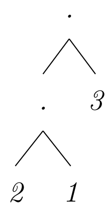
lemma *mera-rotiraj-n*[*simp*]: *mera* ((*rotiraj1* ^^ *n*) *t*) = *mera t* − *n*
  by (*induction n*) *auto*

theorem *dll-rotiraj*: ∃ *n* ≤ *size t*. *dll* ((*rotiraj1* ^^ *n*) *t*)
  by (*induction t*) (*auto simp add*: *mera-0*)

Zadatak 2 *Fold-ovanje nad stablima*

____/ 10 p.

1. *(2p)*
   Definisati tip '*a ldrvo* koji predstavlja binarno stablo koje čuva podatke samo u listovima.
   Definisati instancu *test-drvo* :: *nat ldrvo* tako da predstavlja sledeće drvo



2. *(1p)*

   Definisati funkciju *lkd* :: *'a ldrvo* $\Rightarrow$ *'a list* koja vraća listu elemenata obilaskom stabla metodom levo-koren-desno.

   value *lkd test-drvo* = [*2*,*1*,*3*] — (Ovaj izraz mora da se evaluira u *True*)

3. *(2p)*

   Jedan način foldovanja nad elementima ovog stabla je *fold f* (*lkd d*) *acc*. Definisati funkciju *fold-ldrvo* :: (*'a* $\Rightarrow$ *'b* $\Rightarrow$ *'b*) $\Rightarrow$ *'a ldrvo* $\Rightarrow$ *'b* $\Rightarrow$ *'b* rekurzivno po strukturi drveta koja vraća isti rezultat kao i *fold f* (*lkd d*) *acc*.

   Napomena: Definisanje funkcije *fold-ldrvo* preko *fold* funkcije na iznad naveden način se ne priznaje kao tačno rešenje, već funckiju *fold-ldrvo* morate definisati primitivnom rekurzijom.

   value *fold-ldrvo* (+) *test-drvo 0* = *6* — (Ovaj izraz mora da se evaluira u *True*)

4. *(2p)*

   Pokazati da je *fold f* (*lkd d*) *acc* = *fold-ldrvo f d acc*.

5. *(1p)*

   Definisati funkciju *obrni* :: *'a ldrvo* $\Rightarrow$ *'a ldrvo* koja obrće stablo kao u ogledalu.

   value *ldk* (*obrni test-drvo*) = [*3*,*1*,*2*] — (Ovaj izraz mora da se evaluira u *True*)

6. *(2p)*

   Pokazati da je lkd poredak obrnutog drveta isto što i obrnuti lkd poredak početnog drveta.

Rešenje 2

10 p.

datatype *'a ldrvo* = *List 'a* | *Cvor 'a ldrvo 'a ldrvo*

definition *test-drvo* :: *nat ldrvo* **where**
  test-drvo ≡ *Cvor* (*Cvor* (*List 2*) (*List 1*)) (*List 3*)

primrec *lkd* :: *'a ldrvo* $\Rightarrow$ *'a list* **where**
  lkd (*List x*) = [*x*]
| *lkd* (*Cvor l d*) = *lkd l* @ *lkd d*

value *lkd test-drvo* = [*2*,*1*,*3*]

primrec *fold-ldrvo* :: (*'a* $\Rightarrow$ *'b* $\Rightarrow$ *'b*) $\Rightarrow$ *'a ldrvo* $\Rightarrow$ *'b* $\Rightarrow$ *'b* **where**
  fold-ldrvo f (*List x*) *acc* = *f x acc*
| *fold-ldrvo f* (*Cvor l d*) *acc* = *fold-ldrvo f d* (*fold-ldrvo f l acc*)

value *fold-ldrvo* (+) *test-drvo 0* = *6*

lemma *fold f* (*lkd d*) *acc* = *fold-ldrvo f d acc*
  by (*induction d arbitrary*: *acc*) *auto*

primrec *obrni* :: *'a ldrvo* ⇒ *'a ldrvo* **where**
  obrni (*List x*) = *List x*
| *obrni* (*Cvor l d*) = *Cvor* (*obrni d*) (*obrni l*)

value *lkd* (*obrni test-drvo*) = [*3*,*1*,*2*]

lemma *lkd* (*obrni d*) = *rev* (*lkd d*)
  by (*induction d*) *auto*

Zadatak 3 *Odsecanje liste*

1. *(1p)*
   Definisati funkciju *sadrzi* :: *'a* ⇒ *'a list* ⇒ *bool* koja ispituje da li se element nalazi u listi.

   Sledeća dva izraza se moraju evaluirati u *True*:

   value *sadrzi 5* [*1*,*6*,*2*,*5*,*3*,*4*::*nat*] = *True* — (Sadrži se)

   value *sadrzi 8* [*1*,*6*,*2*,*5*,*3*,*4*::*nat*] = *False* — (Ne sadrži se)

2. *(2p)*

   Pokazati da je funkcija *sadrzi* invarijantna u odnosu na obrtanje liste.

   Savet: Definisati pomoćnu lemu koja opisuje da li se element nalazi u listi kojoj znamo početak i poslednji element.

3. *(1p)*
   Definisati funkciju *razliciti* :: *'a list* ⇒ *bool* koja ispituje da li su svi elementi liste međusobno različiti.

   Sledeća dva izraza se moraju evaluirati u *True*:

   value *razliciti* [*1*,*6*,*2*,*5*,*3*,*4*::*nat*] = *True* — (Svi različiti)

   value *razliciti* [*1*,*4*,*2*,*5*,*2*,*4*::*nat*] = *False* — (Ima istih)

4. *(2p)*

   Pokazati da je funkcija *razliciti* invarijantna na obrtanje liste.

5. *(3p)*
   Pomoću funkcija *fold* i *foldr* definisati funkcije *duzina-fold* i *duzina-foldr*, respektivno, koje računaju dužinu liste.

   Sledeći izrazi se moraju evaluirati u *True*.

   value *duzina-fold* [] = *0* — (Prazna lista)

   value *duzina-fold* [*1*,*2*,*3*::*nat*] = *3* — (Lista dužine 3)

   value *duzina-foldr* [] = *0* — (Prazna lista)

   value *duzina-foldr* [*1*,*2*,*3*::*nat*] = *3* — (Lista dužine 3)

6. *(4p)*

   Pokazati da su *duzina-fold* i *duzina-foldr* korektne, tj. da daju isti rezultat kao i funkcija *length*.

7. *(3p)*

   Definisati funkciju *iseci* :: *'a list* ⇒ *nat* ⇒ *nat* ⇒ *'a list*. Za listu $xs = [x_0, \ldots, x_n]$, poziv *iseci xs s l* vraća listu: $[x_s, \ldots, x_{s+l-1}]$. Ako su vrednosti s i l van opsega funkcija *iseci* vraća kraću ili praznu listu.

   Savet: Koristite opštu rekurziju i *pattern matching*, umesto *if−then−else* izraza. Npr. umesto *f s* = (*if s* = *0 then e1 else e2*) koristiti *f 0* = *e1* i *f s* = *e2*.

   Sledeći izrazi se moraju evaluirati u *True*.

   value *iseci* [*0*,*1*,*2*,*3*,*4*,*5*,*6*::*int*] *2 3* = [*2*,*3*,*4*] — (U opsegu)

   value *iseci* [*0*,*1*,*2*,*3*,*4*,*5*,*6*::*int*] *2 10* = [*2*,*3*,*4*,*5*,*6*] — (Dužina van opsega)

   value *iseci* [*0*,*1*,*2*,*3*,*4*,*5*,*6*::*int*] *10 10* = [] — (Početni indeks van opsega)

8. *(2p)*

   Pokazati da važi: *iseci xs s l1* @ *iseci xs* (*s* + *l1*) *l2* = *iseci xs s* (*l1* + *l2*)

9. *(2p)*

   Pokazati da odsecanje liste zadržava invarijantnost o različitim elementima, tj. ako su elementi liste različiti, onda će biti različiti i elementi isečka.

Rešenje 3

fun *sadrzi* :: *'a* ⇒ *'a list* ⇒ *bool* **where**
  sadrzi a [] = *False*
| *sadrzi a* (*x* # *xs*) = (*a* = *x* ∨ *sadrzi a xs*)

value *sadrzi 5* [*1*,*6*,*2*,*5*,*3*,*4*::*nat*] = *True*
value *sadrzi 8* [*1*,*6*,*2*,*5*,*3*,*4*::*nat*] = *False*

lemma *sadrzi-snoc*[*simp*]: *sadrzi a* (*xs* @ [*x*]) = (*x* = *a* ∨ *sadrzi a xs*)
  by (*induction xs*) *auto*

lemma *sadrzi-rev*: *sadrzi a* (*rev xs*) = *sadrzi a xs*
  by (*induction xs*) *auto*

fun *razliciti* :: *'a list* ⇒ *bool* **where**
  razliciti [] = *True*
| *razliciti* (*x* # *xs*) = (¬ (*sadrzi x xs*) ∧ *razliciti xs*)

value *razliciti* [*1*,*6*,*2*,*5*,*3*,*4*::*nat*] = *True*
value *razliciti* [*1*,*4*,*2*,*5*,*2*,*4*::*nat*] = *False*

lemma *razliciti-snoc*[*simp*]: *razliciti* (*xs* @ [*x*]) = (¬ *sadrzi x xs* ∧ *razliciti xs*)
  by (*induction xs*) *auto*

lemma *razliciti-rev*: *razliciti* (*rev xs*) = *razliciti xs*
  by (*induction xs*) (*auto simp add*: *sadrzi-rev*)

definition *duzina-fold* :: *'a list* ⇒ *nat* **where**
  duzina-fold xs = *fold* (λ -. *Suc*) *xs 0*

definition *duzina-foldr* :: *'a list* ⇒ *nat* **where**
  duzina-foldr xs = *foldr* (λ -. *Suc*) *xs 0*

lemma [*simp*]: *fold* (λ -.*f*) *xs* (*f acc*) = *f* (*fold* (λ-.*f*) *xs acc*)
  by (*induction xs*) (*auto simp add*: *fold-commute-apply*)

lemma *duzina-fold*: *duzina-fold xs* = *length xs*
  unfolding *duzina-fold-def*
  by (*induction xs*) *auto*

lemma *duzina-foldr*: *duzina-foldr xs* = *length xs*
  unfolding *duzina-foldr-def*
  by (*induction xs*) *auto*

fun *iseci* :: *'a list* ⇒ *nat* ⇒ *nat* ⇒ *'a list* **where**
  iseci [] *s l* = []
| *iseci* (*x* # *xs*) *0* (*Suc l*) = *x* # (*iseci xs 0 l*)
| *iseci* (*x* # *xs*) (*Suc s*) *l* = *iseci xs s l*
| *iseci* (*x* # *xs*) *s 0* = []

value *iseci* [*0*,*1*,*2*,*3*,*4*,*5*,*6*::*int*] *2 3* = [*2*,*3*,*4*]
value *iseci* [*0*,*1*,*2*,*3*,*4*,*5*,*6*::*int*] *2 10* = [*2*,*3*,*4*,*5*,*6*]
value *iseci* [*0*,*1*,*2*,*3*,*4*,*5*,*6*::*int*] *10 10* = []

lemma *iseci-append*: *iseci xs s l1* @ *iseci xs* (*s* + *l1*) *l2* = *iseci xs s* (*l1* + *l2*)
  by (*induction xs s l1 rule*: *iseci.induct*) *auto*

lemma *sadrzi-iseci*: *sadrzi x* (*iseci xs s l*) ⟹ *sadrzi x xs*
  by (*induction xs s l rule*: *iseci.induct*) *auto*

lemma *razliciti-iseci*: *razliciti xs* ⟹ *razliciti* (*iseci xs s l*)
  by (*induction xs s l rule*: *iseci.induct*) (*auto simp add*: *sadrzi-iseci*)