Prvi praktični kolokvij - Haskell

Matematička logika u računarstvu

20. travnja 2016.

Ovo je "*open book*" kolokvij. Dozvoljeno je korištenje bilo kakvih materijala — bilješke s vježbi, Haskell help, Hoogle, tutoriali, postovi na online forumima,... — **nastalih prije** kolokvija (npr. dozvoljeno je na *StackOverflow*u naći rješenje nekog zadatka, ali nije dozvoljeno tamo postaviti pitanje kako se rješava neki zadatak). Također, nije dozvoljena komunikacija (razgovor, chat, razmjena bilježaka) **među** studentima.

Potrebne / korisne "hint-datoteke" nalaze se u Github repozitoriju, `mlr/Haskell/Kolokvij`. Rješenja zadataka pišite svako u svoju datoteku (`Z`*`n`*`.hs`, gdje je *n* broj zadatka), te na kraju sve datoteke pošaljite mailom na `veky@math.hr`. Očekuje se da će svaka datoteka s rješenjem (osim `Z4.hs`) nastati dodavanjem ili editiranjem koda u nekoj od hint-datoteka. Svaka datoteka mora biti sama sebi dovoljna, i sama za sebe se mora moći učitati u GHC bez warninga i grešaka. Korisno je u datoteku uključiti testove iz kojih se vidi kako funkcije rade (bilo kao komentare, ili kao varijable nazvane `test1`, `test2`, ...).

Predviđeno vrijeme rješavanja je 150 minuta. Svaki zadatak vrijedi 5 bodova. Postoji i jedan bonus zadatak. Maksimalni broj bodova koji se može osvojiti je 25. Rezultati i uvidi će biti u ponedjeljak 2. svibnja u 12 sati. Zadaci su s druge strane papira. Sretno!

Veky

1. a) Napišite funkciju `hasSeqRep` koja za formulu logike sudova vraća ima li ona nizovnu reprezentaciju (je li najveća mjesnost veznika u toj formuli jednaka 2).
   b) Napišite funkciju `newAtom`, koja za (konačnu) listu formula logike sudova vraća neki atom koji se ne pojavljuje ni u jednoj od formula u listi.
2. a) Među veznike dodajte i "ternarni operator" poznat i kao "?:" iz Ca. Ako njegovu istinitosnu funkciju označimo s `t`, tada `t[x,y,z]` treba biti ekvivalentno `y` ako je `x True`, a `z` ako je `x False`. Dodajte i ispis, tako da se `Compound ternary_c [a, b, c]` (gdje je `ternary_c = Conn 3 t`) ispisuje kao `(a ? b : c)`.
   b) Implementirajte funkciju `fullTable :: Connective -> [[Bool]]`, koja daje "pravu" tablicu istinitosti (ne samo njen zadnji stupac kao funkcija `table`) kao listu listi, gdje vanjska lista ima 2^k elemenata, a unutarnje imaju $k+1$ elemenata.
3. a) Napišite funkciju `tally`, koja za formulu logike prvog reda vraća uređeni par (S,V), gdje je S skup (`Data.Set`) slobodnih varijabli, a V skup vezanih varijabli. Pazite: skupovi nisu nužno disjunktni, recimo $p(y) \wedge \exists y q(y)$ ima y i kao slobodnu i kao vezanu varijablu.
   b) Za formule logike prvog reda implementirajte funkciju `pNeg :: Formula -> Formula` koja "propagira negaciju", odnosno negira formulu prema pravilima za negiranje kvantifikatora i veznika. Ako je veznik u složenoj formuli jedan od preddefiniranih, treba upotrijebiti poznata pravila za negaciju veznika (recimo, `pNeg (f-->g)` je `(f & pNeg g)`, a `pNeg (ne f)` je samo `f`), a ako nije, treba na licu mjesta napraviti novi veznik (iste mjesnosti) koji ima negiranu istinitosnu funkciju.
4. Na goo.gl/hed6n9 se nalazi formalna definicija regularnih izraza. Definirajte tip `RegEx` u Haskellu pomoću `data` naredbe (sami smislite tagove). Instancirajte `Show RegEx` na odgovarajući način (pokušajte ne koristiti zagrade gdje očito ne treba).
   Definirajte funkciju `exactly :: String -> RegEx`, koja stvara (konkatenacijom) regularni izraz koji prepoznaje točno zadani string. Definirajte i rekurzivnu funkciju `singular :: RegEx -> Bool`, koji vraća prepoznaje li zadani regularni izraz praznu riječ (ovo možete napraviti strukturalnom analizom regularnog izraza, ne trebate raditi općeniti algoritam za prepoznavanje riječi).
5. Popravite problem "slučajnog vezanja" varijabli u implementaciji lambda-računa koju smo napravili na predavanjima. U hint-datoteci je već prepravljen tip Slovo tako da ima beskonačno mnogo elemenata. Uz taj popravak, `(λxλy.x+y)y` se treba β-reducirati u `λy'.y+y'`. Pazite da doista stvorite "svježu" varijablu, čak i ako je varijabla s crticom već u dosegu.
6. (Bonus) Definirajte klauzule i klauzalne forme u logici sudova (recimo `data Clause = BackArrow (Set Atom) (Set Atom)`). Implementirajte rezoluciju kao funkciju `contradictory :: ClausalForm -> Bool`. Pripazite da algoritam staje čak i kad (konačna) klauzalna forma nije proturječna.