Lista zadan nr 5- rozwiazania

Lukasz Kopyto

March 26, 2024

1 Zadanie 1.

1.1 Tresc:

Przypomnij sobie definicje funkcji map. Nastepnie pokaz, ze dla dowolnych funkcji f i g oraz listy xs zachodzi map f (map g xs) ≡ map (fun x → f (g x)) xs Mozesz zalozyc, ze funkcje f i g poprawnie obliczaja sie do wartosci dla dowolnego argumentu.

1.2 Rozwiazanie

Definicja funkcji map:

```
let rec map f xs = match xs with
    | [] -> []
    | h :: t -> (f h) :: (map f t)
```

Zasada indukcji dla list typu 'a list

Dla kazdej wlasnosci P, jesli zachodzi P([ ]) oraz dla kazdego a : 'a, as : 'a list P(as) impikuje P(a :: as), to dla kazdej listy as zachodzi P(as)

Przeprowadzimy dowod przez indukcje dla list. Ustalmy dowolne funkcj f i g oraz dowolna liste xs.

Podstawa indukcji:

L = map f (map g [ ]) = z def. map = map f [ ] = z def. map = [ ]

P = map (fun x → f (g x)) [ ] = z def. map = [ ]

Zatem podstawa indukcji zachodz i.

Krok indukcyjny:

Wezmy dowolne xs i zalozmy ze map f (map g xs) ≡ map (fun x → f (g x)) xs. Pokaze, ze dla kazdego a : 'a map f (map g a::xs) ≡ map (fun x → f (g x)) a :: xs

L = map f (map g a::xs) = z def. map = map f ((g a) :: (map g xs)) = z def. map = f (g a) :: map f (map g xs) = zal indukcyjne = f (g a) :: map (fun x → f (g x)) xs

P = map (fun x → f (g x)) (a :: xs) = f (g a) :: (map (fun x → f (g x)) xs)
Zatem L = P. Na mocy zasady indukcji dla list zachodzi teza.

2 Zadanie 2.

2.1 Tresc:

Pokaz ze funkcja append zawsze wylicza sie do wartosci. Tzn. pokaz ze dla dowolnych list xs, ys istnieje lista zs taka ze append ys xs ≡ zs.

2.2 Rozwiazanie:

Definicja appenda:

```
let rec append xs ys = match xs with
    | [] -> ys
    | h :: t -> h :: append t ys
```

Ustalmy dowolna liste ys
Przeprowadzimy dowod przez indukcje wzgledem dlugosci listy xs.
Krok indukcyjny:
Dla xs = [ ], niech zs = ys.
Wtedy append xs ys = append [ ] ys = z def. append = ys
Krok indukcyjny:
Ustalmy dowolna liste xs i zalozmy ze append ys xs = zs. Pokaze, ze dla kazdego a : 'a oraz ys : 'a list, append a::ys xs = a :: zs
L = append a::ys xs = z def. appenda = a :: append ys xs = a :: zs = P
Zatem L = P, czyli na mocy zasady indukcji dla list, zachodzi teza.

3 Zadanie 3.

3.1 Tresc:

Formuly w negacyjnej postaci normalnej(nnf) mozna opisac nastepujacym typem danych, sparametryzowanym typem opisujacym zmienne.

```
type 'v nnf =
    | NNFLit of bool * 'v
    | NNFConj of 'v nnf * 'v nnf
    | NNFDisj of 'v nnf * 'v nnf
```

Flaga boolowska w konstruktorze literalu oznacza, czy zmienna jest zanegowana(wartosc true), czy nie (wartosc false). Sformuluj zasade indukcji dla typu NNF.

3.2 Rozwiazanie:

Zasada indukcji dla typu 'v nnf

Dla kazdej wlasnosci P, jesli dla dowolnych (b : bool), (v : 'v) zachodzi P(NNLit(b, v)) oraz dla kazdego (n1 : 'v nnf), (n2 : 'v nnf), P(n1) i P(n2) implikuje P(NNFConj(n1, n2)) oraz P(n1) i P(n2) implikuje P(Conj(n1, n2)), to wtedy dla kazdej formuly f zapisanej w 'v nnf zachodzi P(f).

4 Zadanie 4.

4.1 Tresc:

Zdefiniuj funkcje neg_nnf : 'v nnf $\rightarrow$ 'v nnf negujaca formule zapisana w negacyjnej postaci normalnej. Nastepnie pokaz, ze neg_nnf (neg_nnf ϕ) $\equiv \phi$

4.2 Rozwiazanie:

Funkcja:

```
let rec neg_nnf = function
    | NNFLit (x, v) -> NNFLit(not x, v)
    | NNFConj (a, b) -> NNFDisj((neg_nnf a), (
       neg_nnf b))
    | NNFDisj (a, b) -> NNFConj((neg_nnf a), (
       neg_nnf b))
```

Dowod przez indukcje:

Baza indukcji:

Ustalmy dowolny literal l = NNFLit(b:bool, v). wtedy:

L = neg_nnf (neg_nnf l) = neg_nnf (neg_nnf NNFLit(b, v)) = z def neg_nnf = neg_nnf NNFLit(not b, v) = z def neg_nnf = NNFLit(not not b, v) = NNFLit(b,v) = l = P.

Krok indukcyjny: Ustalmy dowolne formuly ($\phi : v'$ nnf) i ($\psi : v'$ nnf).

Zalozmy, ze zachodzi neg_nnf (neg_nnf ϕ) $\equiv \phi$ oraz neg_nnf (neg_nnf ψ) $\equiv \psi$.

Pokaze ze zachodzi neg_nnf (neg_nnf NNFConj(ψ, ϕ)) oraz neg_nnf (neg_nnf NNFDisj(ϕ, ψ)).

1. neg_nnf (neg_nnf NNFConj(ψ, ϕ)) = z def neg_nnf = neg_nnf NNFDisj(neg_nnf ψ, neg_nnf ϕ) = z def neg_nnf = NNFConj(neg_nnf (neg_nnf ψ), neg_nnf (neg_nnf ϕ) = zal indukcyjne = NNFConj(ψ, ϕ))

2. neg_nnf (neg_nnf NNFDisj(ψ, ϕ)) = z def neg_nnf = neg_nnf NNFConj(neg_nnf ψ, neg_nnf ϕ) = z def neg_nnf = NNFDisj(neg_nnf (neg_nnf ψ), neg_nnf (neg_nnf ϕ) = zal indukcyjne = NNFDisj(ψ, ϕ))

Zatem na mocy zasady indukcji, teza zachodzi.

5 Zadanie 5

5.1 Tresc:

Zdefiniuj funkcje eval_nnf interpretujaca formule w negacyjnej postaci normalnej, przy zadanym wartosciowaniu zmiennych. Nastepnie pokaz ze dla dowolnej formuly ϕ i wartosciowania σ zachodzi eval_nnf σ (neg_nnf ϕ) $\equiv$ not (eval_nnf σ ϕ)

5.2 Rozwiazanie:

```
let rec eval_nnf sigma phi =
  match phi with
  | NNFLit (x, v) -> let t = sigma v in if x then not
     t else t
  | NNFConj (v1, v2) -> (eval_nnf sigma v1) && (
     eval_nnf sigma v2)
  | NNFDisj (v1, v2) -> (eval_nnf sigma v1) || (
     eval_nnf sigma v2)
```

Ustalmy dowolne wartosciowanie σ oraz dowolna formule ϕ zapisana w NNF. Dowod przez indukcje strutkturalna wzgledem struktury typu 'v nnf.

Podstawa indukcji:

Dla ϕ bedacego literalem, tzn ϕ = NNFLit(b:bool, v) mamy:

L = eval_nnf σ (neg_nnf ϕ) = eval_nnf σ (neg_nnf NNFLit (b, v)) = eval_nnf σ NNFLit (not b, v) = mamy dwa przypadki:

1. b = true. Wtedy eval_nnf σ NNFLit (not b, v) = eval_nnf σ NNFLit (false, v) = sigma v
2. b = false. Wtedy eval_nnf σ NNFLit (not b, v) = eval_nnf σ NNFLit (true, v) = not (sigma v)

P = not (eval_nnf σ ϕ) = not (eval_nnf σ NNFLit(b, v)) = tutaj tez mamy dwa przypadki:

1. b = true. Wtedy not (eval_nnf σ NNFLit(b, v)) = not (eval_nnf σ NNFLit(false, v)) = not (not (sigma v)) = sigma v
2. b = false. Wtedy not (eval_nnf σ NNFLit(b, v)) = not (eval_nnf σ NNFLit(false, v)) = not (sigma v)

W obu przypadkach otrzymalismy, ze L = P. Zatem baza indukcji zachodzi.

Ustalmy dowolne wartosciowanie σ oraz formuly ϕ, ψ. Zalozmy ze:

(1) eval_nnf σ (neg_nnf ϕ) $\equiv$ not (eval_nnf σ ϕ).

(2) eval_nnf σ (neg_nnf ψ) $\equiv$ not (eval_nnf σ ψ).

Pokaze ze zachodzi:

(1) eval_nnf σ (neg_nnf NNFConj(ϕ, ψ)) $\equiv$ not (eval_nnf σ NNFConj(ϕ, ψ)).

(2) eval_nnf σ (neg_nnf NNFDisj(ϕ, ψ)) $\equiv$ not (eval_nnf σ NNFDisj(ϕ, ψ)).

NNFConj

L = eval_nnf σ (neg_nnf NNFConj(ϕ, ψ)) = z def neg_nnf = eval_nnf σ NNFDisj((neg_nnf ϕ), (neg_nnf ψ)) = z def eval_nnf = (eval_nnf $\sigma\phi$) —— (eval_nnf $\sigma\psi$) (z zal ind) $\equiv$ (not (eval_nnf σ))

6 Zadanie 6

6.1 Tresc:

Formuly rachunku zdan mozemy opisac nastepujacym typem:

```
type 'v formula =
    | Var of 'v
    | Neg of 'v formula
    | Conj of 'v formula * 'v formula
    | Disj of 'v formula * 'v formula
```

Zdefiniuj funkcja to_nnf transformujaca formule do rownowaznej formuly w negacyjnej postaci normalnej. Mozesz zdefiniowac funkcje pomocnicze, ale wszystkie funkcje(wzajemnie) rekurencyjne powinny byc strukturalnie rekurencyjne.

6.2 Rozwiazanie:

```
let rec to_nnf phi = match phi with
  | Var v -> NNFLit (false, v)
  | Neg psi -> neg_nnf (to_nnf psi)
  | Conj (c1, c2) -> NNFConj( (to_nnf c1), (to_nnf c2)
     )
  | Disj (d1, d2) -> NNFDisj( (to_nnf d1), (to_nnf d2)
     )
```

7 Zadanie 7.

7.1 Tresc:

Zdefiniuj funkcje eval_formula interpretujaca formuly z poprzedniego zadania. Nastepnie pokaz ze: eval_nnf σ (to_nnf ϕ) $\equiv$ eval_formula σ ϕ

7.2 Rozwiazanie

Funkcja eval_formula:

```
let rec eval_formula sigma phi = match phi with
    | Var v -> sigma v
    | Neg psi -> not (eval_formula sigma psi)
    | Conj (c1, c2) -> (eval_formula sigma c1) && (
       eval_formula sigma c2)
    | Disj (d1, d2) -> (eval_formula sigma d1) || (
       eval_formula sigma d2)
```

Zasada indukcji dla typu 'v formula

Dla kazdej wlasnosci P, jesli dla dowolnego (x : v') zachodzi P(Var x) oraz dla kazdego (ϕ : 'v formula), (ψ : 'v formula) zachodzi:

- P(Neg ϕ)
- $P(\phi) \wedge P(\psi) \implies P(\text{Conj}(\phi, \psi))$
- $P(\phi) \wedge P(\psi) \implies P(\text{Disj}(\phi, \psi))$

To wtedy dla kazdej formuly ϕ typu 'v formula zachodzi $P(\phi)$

Dowod przez indukcje struktutralna, ze eval_nnf σ (to_nnf ϕ) $\equiv$ eval_formula σ ϕ.

Ustalmy dowolne wartosciowanie σ.

Baza indukcji. Dla ($\phi : v'$):

L = evalnnf sigma (tonnf phi) = evalnnf sigma NNFlit(false, phi) = sigma phi

P = evalformula sigma phi = sigma phi

L = P zatem baza indukcji zachodzi.

Ustalmy dowolne formuly typu phi, psi typu 'v formula i zalozmy ze

(i) eval_nnf σ (to_nnf ϕ) $\equiv$ eval_formula σ ϕ

(ii) eval_nnf σ (to_nnf ψ) $\equiv$ eval_formula σ ψ.

Pokaze ze zachodzi tez:

(a) eval_nnf σ (to_nnf Conj(ϕ, ψ)) $\equiv$ eval_formula σ Conj(ϕ, ψ).

(b) eval_nnf σ (to_nnf Disj(ϕ, ψ)) $\equiv$ eval_formula σ Disj(ϕ, ψ).

(c) eval_nnf σ (to_nnf Neg(ϕ)) $\equiv$ eval_formula σ Neg(ϕ).

(a) L = eval_nnf σ (to_nnf Conj(ϕ, ψ)) = z def to_nnf = eval_nnf σ NNFConj((to_nnf ϕ), (to_nnf ψ)) = z def eval_nnf = (eval_nnf σ (to_nnf ψ)) && (eval_nnf σ (to_nnf ϕ)) $\equiv$ zal ind (eval_formula σ ϕ) && (eval_formula σ ψ) = z def eval_formula = eval_formula σ Conj(ϕ, ψ) = P

(b) L = eval_nnf σ (to_nnf Disj(ϕ, ψ)) = z def to_nnf = eval_nnf σ NNFDisj((to_nnf ϕ), (to_nnf ψ)) = z def eval_nnf = (eval_nnf σ (to_nnf ψ)) $\|$ (eval_nnf σ (to_nnf ϕ)) $\equiv$ zal ind (eval_formula σ ϕ) $\|$ (eval_formula σ ψ) = z def eval_formula = eval_formula σ Disj(ϕ, ψ) = P

(c) L = eval_nnf σ (to_nnf Neg(ϕ)) = z def to_nnf = eval_nnf σ (neg_nnf (to_nnf ϕ))$\equiv$ z lematu $\equiv$ not (eval_nnf σ (to_nnf ϕ)

P = eval_formula σ Neg(ϕ) = z def eval_formula = not (eval_formula σ ϕ) $\equiv$ zal ind $\equiv$ not (eval_nnf σ (to_nnf ϕ))

Czyli L $\equiv$ P. Zatem na mocy zasady indukcji, teza zachodzi.

8 Zadanie 8.

8.1 Tresc:

8.2 Rozwiazanie: