Programowanie Funkcyjne 2020

Lista zadań nr 5 dla grup mabi, mbu, ppo i efes

Na zajęcia 3 i 4 listopada 2020

Zadanie 1 (3p). Rozważmy następującą reprezentację drzew binarnych z etykietami w wierzchołkach wewnętrznych:

```
type 'a btree = Leaf | Node of 'a btree * 'a * 'a btree
```

Powiemy że drzewo jest *zbalansowane*, jeśli w każdym jego wierzchołku `t` liczby wierzchołków w lewym i prawym poddrzewie `t` różnią się co najwyżej o 1.

1. Zaimplementuj efektywną procedurę sprawdzającą czy dane drzewo jest zbalansowane.
2. Zaimplementuj procedurę, która dla danej listy etykiet `xs` zbuduje zbalansowane drzewo `t`, takie że `xs` jest listą etykiet `t` w porządku preorder.

Zadanie 2 (5p). Kiedy wykonujemy modyfikacje struktury danych, często chcemy wykonać zmiany niedaleko siebie. W przypadku trwałych struktur danych powoduje to narzut związany z odtwarzaniem całej struktury danych po każdej zmianie. Możemy ten narzut istotnie zmniejszyć wprowadzając dodatkową strukturę danych, która pozwoli nam reprezentować *miejsca* w początkowej strukturze i oddzielić nawigację od jej modyfikacji — a w konsekwencji uniknąć narzutu związanego z każdorazowym odbudowywaniem struktury.

1. Zdefiniuj typ danych `'a place` reprezentujący miejsca na liście.
2. Zdefiniuj funkcje `findNth : 'a list -> int -> 'a place` i `collapse : 'a place -> 'a list`, odpowiednio znajdującą n-te miejsce na danej liście i zapominającą informację o miejscu (tj. odbudowującą początkową strukturę danych).
3. Zdefiniuj funkcje `add : 'a -> 'a place -> 'a place` i `del : 'a place -> 'a place`, odpowiednio dodającą i usuwającą element w miejscu listy reprezentowanym przez argument. Obie funkcje powinny działać w czasie stałym i spełniać poniższe równości:

   ```
   collapse (add 5 (findNth [1; 2; 3; 4] 2)) = [1; 2; 5; 3; 4]
   collapse (del (findNth [1; 2; 3; 4] 2)) = [1; 2; 4]
   del (add x p) = p
   ```

   dla dowolnych `x : 'a` i `p : 'a place`
4. Zdefiniuj funkcje służące do nawigacji, `next` i `prev`, obie typu `'a place -> 'a place`, przechodzące odpowiednio do następnego i poprzedniego miejsca na liście, i działające w czasie stałym.
5. Listy nie są jedynymi strukturami danych dla których możemy rozważać miejsca pozwalające na modyfikację i drobnoziarnistą nawigację: zaproponuj typ `'a btplace` pozwalający reprezentować miejsca w drzewach binarnych z poprzedniego zadania. Nie musisz implementować funkcji modyfikacji czy nawigacji, ale dobrze zastanowić się nad odpowiednim *interfejsem*.

Zadanie 3 (1p). W tym oraz następnych zadaniach zajmiemy się fragmentem logiki intuicjonistycznej, w którym formuły składają się tylko ze zmiennych zdaniowych (p,q,r,$\dots$), binarnego spójnika implikacji ($\to$) oraz 0-arnego spójnika fałszu ($\bot$).[1] Poprawnymi formułami są na przykład:

$$p \qquad\quad p \to q \qquad\quad (p \to \bot) \to \bot \qquad\quad (p \to q \to r) \to (p \to q) \to p \to r.$$

Wypisując formuły przyjmujemy, że implikacja wiąże w prawo, więc ta ostatnia formuła w istocie oznacza

$$(p \to (q \to r)) \to ((p \to q) \to (p \to r)).$$

[1] Rozszerzenie tego rachunku do pełnej logiki intuicjonistycznej, z koniunkcją, dysjunkcją i stałą prawdy nie wprowadza istotnych utrudnień — ale dodaje trochę kodu do napisania. Jeśli masz ochotę, rozszerz logikę o dodatkowe konstrukcje!

Na stronie przedmiotu znajduje się szablon rozwiązania. W plikach `logic.mli` oraz `logic.ml` uzupełnij definicję typu `formula` o zaproponowaną przez siebie reprezentacją formuł w rozważanej logice.

Wskazówka: Zauważ, że formuły to nic innego, jak drzewa binarne, które mają dwa rodzaje liści

Zadanie 4 (2p). Uzupełnij definicję funkcji `string_of_formula` znajdującą się w pliku `logic.ml`. Funkcja powinna przekształcać formułę w napis czytelny dla człowieka, używający jak najmniejszej liczby nawiasów. Funkcja ta jest prywatna dla modułu `Logic` i służy jedynie zaimplementowaniu funkcji `pp_print_formula`, którą można zarejestrować w interpreterze, jako sposób wyświetlania formuł:

```
utop # #install_printer Logic.pp_print_formula ;;
utop # let f = {zależne od Twojej reprezentacji formuł} ;;
val f : Logic.formula = (p → ⊥) → p → q
```

Uwaga: Jeśli jesteś odważny, możesz rozwiązać trudniejszy wariant tego zadania, polegający na bezpośrednim zaimplementowaniu funkcji `pp_print_formula`. W tym celu zapoznaj się z modułem `Format` z biblioteki standardowej. Za rozwiązanie które ładnie formatuje bardzo duże formuły, można dostać dodatkowe punkty!

Zadanie 5 (3p). Wprowadzimy teraz formalny system dowodzenia do naszej logiki. Będzie to wariant systemu naturalnej dedukcji, znanego Wam z kursu logiki. *Osądem* nazwiemy parę $\Gamma \vdash \varphi$, taką że Γ jest skończonym zbiorem formuł (zwanym *założeniami*), a φ jest formułą (zwaną *tezą* albo *konsekwencją*). Znak $\vdash$ pełni tu tylko rolę przecinka. Dowody są drzewami, zbudowanymi z następujących reguł wnioskowania.

$$\frac{}{\{\varphi\} \vdash \varphi}(\mathrm{Ax}) \qquad \frac{\Gamma \vdash \varphi}{\Gamma \setminus \{\psi\} \vdash \psi \to \phi}(\to\mathrm{I}) \qquad \frac{\Gamma \vdash \varphi \to \psi \quad \Delta \vdash \varphi}{\Gamma \cup \Delta \vdash \psi}(\to\mathrm{E}) \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}(\bot\mathrm{E})$$

Innymi słowy, dowody to drzewa, których wszystkie wierzchołki są etykietowane osądami i spełniają jeden z warunków:

- wierzchołek nie ma dzieci i jest etykietowany osądem $\{\varphi\} \vdash \varphi$, dla pewnej formuły φ;
- wierzchołek ma tylko jedno dziecko, etykietowane osądem $\Gamma \vdash \varphi$, zaś samo jest etykietowane osądem $\Gamma \setminus \{\psi\} \vdash \psi \to \phi$;
- wierzchołek ma dwójkę dzieci, których etykiety mają postać odpowiednio $\Gamma \vdash \varphi \to \psi$ oraz $\Delta \vdash \varphi$ (dla pewnych Γ, Δ, φ oraz ψ), zaś sam wierzchołek ma etykietę $\Gamma \cup \Delta \vdash \psi$;
- wierzchołek ma etykietę $\Gamma \vdash \phi$ oraz tylko jedno dziecko o etykiecie $\Gamma \vdash \bot$.

Osąd ma dowód, jeśli istnieje poprawne drzewo dowodu, którego korzeń jest etykietowany tym osądem. Osądy mające dowód nazywamy *twierdzeniami*.

Zaproponuj typ danych reprezentujący twierdzenia w rozważanej logice i uzupełnij jego definicję w pliku `logic.ml`. Nie wpisuj jej w pliku `logic.mli`! Wtedy na zewnątrz modułu `Logic` będzie można konstruować dowody tylko za pomocą funkcji dostarczonych przez ten moduł — typ twierdzeń stanie się *abstrakcyjny*. Przypomnij sobie pojęcie *abstrakcji danych* i zastanów się jak zapewnić, że nie da się skonstruować niepoprawnego dowodu. Czy reprezentacja twierdzeń powinna zawierać informację o dowodzie?

Dodatkowo zaimplementuj funkcje `assumptions` oraz `consequence` zwracające odpowiednio założenia i tezę twierdzenia. Pozwolą one na zarejestrowanie funkcji drukującej twierdzenia w interpreterze.

```
utop # #install_printer Logic.pp_print_theorem ;;
```

Zadanie 6 (4p). Zaimplementuj funkcje `by_assumption`, `imp_i`, `imp_e` oraz `bot_e` z modułu `Logic` odpowiadające regułom wnioskowania. Ich dokładną specyfikację znajdziesz w pliku `logic.mli`

Zadanie 7 (2p). Korzystając z modułu `Logic` zbuduj dowody następujących twierdzeń:

- $\vdash p \to p$,
- $\vdash p \to q \to p$,
- $\vdash (p \to q \to r) \to (p \to q) \to p \to r$,
- $\vdash \bot \to p$.