Programowanie Funkcyjne 2018

Lista zadań nr 5

14 listopada 2018

Zadanie 1 (3 pkt). Zdefiniuj typ do reprezentacji leniwych nieskończonych ciągów elementów dowolnego typu (strumieni) tak, jak na wykładzie. Następnie:

- zdefiniuj strumień przybliżający wartość liczby π z rosnącą dokładnością, korzystając ze wzoru Leibniza:
$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$
- napisz funkcję, która dla danej funkcji f przekształca dowolny strumień $x_1, x_2, x_3, \dots$ w strumień postaci $f\, x_1\, x_2\, x_3, f\, x_2\, x_3\, x_4, f\, x_3\, x_4\, x_5, \dots$
- zastosuj funkcję z poprzedniego punktu aby stworzyć nowy strumień, zbiegający znacznie szybciej do liczby π, przy użyciu transformacji Eulera:
$$F\, x\, y\, z = z - \frac{(y-z)^2}{x - 2y + z}$$
i strumienia z punktu pierwszego

Zapisz te same definicje używając konstrukcji `lazy` i `force` z modułu `Lazy`. Dlaczego `lazy` jest specjalną konstrukcją języka, a nie funkcją?

Zadanie 2 (9 pkt). Mamy ciąg szklanek o znanych pojemnościach, a także kran i zlew. W każdym ruchu możemy wykonać jedną z trzech czynności:

- napełnić jedną ze szklanek wodą z kranu (`FILL`),
- opróżnić jedną ze szklanek do zlewu (`DRAIN`),
- przelać wodę z jednej ze szklanek do innej (`TRANSFER`).

Stan takiego układu szklanek możemy zapisać przy użyciu dwóch list reprezentujących odpowiednio pojemności szklanek i ilość wody znajdującą się w każdej z nich. Przykładowo, para `([4; 9], [4; 6])` oznacza że dysponujemy dwiema szklankami, oraz że pierwsza z nich jest pełna, zaś w drugiej znajduje się 6 jednostek wody. W takim przypadku efekty poniższych ruchów następująco zmieniają zawartość szklanek (czyli drugą z powyższych list; zwróć uwagę że można napełnić pełną szklankę):

- `FILL 1` $\rightarrow$ `[4; 9]`
- `FILL 0` $\rightarrow$ `[4; 6]`
- `DRAIN 0` $\rightarrow$ `[0; 6]`
- `TRANSFER (0, 1)` $\rightarrow$ `[1; 9]`.

Dla danego zestawu szklanek i danej objętości wody, rozwiązaniem nazywamy ciąg ruchów, który prowadzi do uzyskania w dowolnej ze szklanek zadanej objętości wody. Przykładowo, dla szklanek `[4, 9]` i objętości `5`, rozwiązaniem jest ciąg `[FILL 1, TRANSFER (1, 0)]` (a także wiele innych, redundantnych ciągów).

Użyj leniwych list aby zdefiniować funkcję `nsols : (int list * int) -> int -> move list list`, taką że `nsols (glasses, volume) n` zwróci listę `n` najkrótszych rozwiązań problemu zadanego przez `glasses` i `volume` (typ danych move powinien reprezentować pojedynczy ruch). W przypadku gdy rozwiązanie nie istnieje, program może się zapętlić.

Wskazówka: Ponieważ dopuszczamy redundantne ciągi (np. przelewanie z pustego w próżne), a także możemy się zapętlić gdy rozwiązanie nie istnieje, nie potrzeba sprawdzać czy daną konfigurację można osiągnąć w inny (potencjalnie lepszy) sposób.

Uwaga: Dzięki użyciu leniwych list, możliwa jest implementacja w której w programie

```
let f = nsols (g, v) in (f n; f n)
```

drugie wywołanie `f n` działa w czasie O(n).

Dedukcja naturalna w rachunku zdań

W poniższych zadaniach zajmiemy się systemem dedukcji naturalnej dla rachunku zdań (który możecie pamiętać z Logiki dla Informatyków). Systemem tym będziemy się zajować również w przyszłym tygodniu; zadania z tego tygodnia mają na celu zaznajomienie z systemem żeby ułatwić dalszą pracę za tydzień.

$$\frac{P \quad Q}{P \wedge Q}\ \textsc{andI} \qquad \frac{\boxed{\begin{array}{c}P\\ \vdots\\ Q\end{array}}}{P \Rightarrow Q}\ \textsc{impI} \qquad \frac{P}{P \vee Q}\ \textsc{orI}_1 \qquad \frac{Q}{P \vee Q}\ \textsc{orI}_1 \qquad \frac{}{\top}\ \textsc{trueI}$$

$$\frac{P \wedge Q}{P}\ \textsc{andE}_1 \qquad \frac{P \wedge Q}{Q}\ \textsc{andE}_2 \qquad \frac{P \quad P \Rightarrow Q}{Q}\ \textsc{impE} \qquad \frac{P \vee Q \qquad \boxed{\begin{array}{c}P\\ \vdots\\ R\end{array}} \qquad \boxed{\begin{array}{c}Q\\ \vdots\\ R\end{array}}}{R}\ \textsc{orE} \qquad \frac{\bot}{P}\ \textsc{falseE}$$

Rysunek 1: Reguły wnioskowania w dedukcji naturalnej dla rachunku zdań

Dowody twierdzeń w rozważanym systemie są zbudowane przy użyciu reguł wnioskowania przedstawionych na Rys. 1, gdzie wielkie litery oznaczają *zmienne schematyczne*, tj. w każdym wystąpieniu reguły w konkretnym dowodzie każda zmienna schematyczna oznacza pewną formułę rachunku zdań. Przykładowo, dowód tautologii $p \wedge (p \Rightarrow q) \Rightarrow q$ możemy zapisać następująco:

$$\frac{\boxed{\begin{array}{l}p \wedge (p \Rightarrow q)\\ \qquad \dfrac{\dfrac{p \wedge (p \Rightarrow q)}{p}\ \textsc{andE}_1 \qquad \dfrac{p \wedge (p \Rightarrow q)}{p \Rightarrow q}\ \textsc{andE}_2}{q}\ \textsc{impE}\end{array}}}{p \wedge (p \Rightarrow q) \Rightarrow q}\ \textsc{impI}$$

W pierwszej regule, wprowadzania implikacji, przyjmujemy tu $P = p \wedge (p \Rightarrow q)$ i $Q = q$, i podobnie w pozostałych regułach, zaś w liściach tak powstałego drzewa możemy zakończyć dowód ponieważ formuła którą pragniemy uzyskać jest tą samą, która znajduje się w ramce wprowadzonej przez wprowadzanie implikacji.

Zadanie 3 (4 pkt). Na razie nie będziemy zajmować się sprawdzaniem czy dany dowód jest poprawny. Zauważmy jednak już teraz, że o ile drzewa dowodu są (w miarę) czytelne dla człowieka, nie są specjalnie wygodne dla programisty. Wygodniej i znacznie bardziej liniowo, choć prawdopodobnie mniej czytelnie dla człowieka możemy zapisać powyższy dowód następująco:

```
[p ∧ (p ⇒ q) :
  p;
  p ⇒ q;
  q];
p ∧ (p ⇒ q) ⇒ q
```

W powyższym zapisie abstrahujemy od tego których konkretnie reguł używamy żeby wyprowadzić poszczególne formuły, a geometryczne ramki klasycznego zapisu zapisujemy przy użyciu nawiasów kwadratowych.

Zamodeluj powyższy sposób zapisu dowodów przy użyciu typów danych dostępnych w OCamlu. **Uwaga**: to jest *ważne* zadanie. Poświęć trochę czasu żeby zastanowić się w jaki sposób dowody są sformułowane, żeby jak najwierniej odwzorować je przy pomocy swojej reprezentacji. Nie wahaj się wrócić do niego i zmienić reprezentacji biorąc pod uwagę wnioski z kolejnego zadania.

Zadanie 4 (6 pkt). Wystąpienia zmiennych w formule logicznej φ możemy podzielić na *pozytywne* i *negatywne* w następujący sposób:

- jeśli formuła jest zmienną ($\varphi = p$), to p występuje w niej pozytywnie, a żadna zmienna nie występuje w niej negatywnie;
- jeśli formuła jest koniunkcją ($\varphi = \varphi_1 \wedge \varphi_2$), to wszystkie wystąpienia zmiennych zachowują swój „znak": wystąpienia pozytywne w φ_1 lub φ_2 są pozytywnymi wystąpieniami w φ, i analogicznie dla wystąpień negatywnych;
- dysjunkcja zachowuje „znak" analogicznie do koniunkcji
- jeśli formuła jest implikacją ($\varphi = \varphi_1 \Rightarrow \varphi_2$), sytuacja jest ciekawsza. Zmienne które występują pozytywnie w φ_2 występują pozytywnie w φ (negatywne analogicznie), ale zmienne z φ_1 „zmieniają znak": jeśli x występuje pozytywnie w φ_1, to występuje *negatywnie* w φ (a jeśli występuje negatywnie w φ_1, to wystąpienie to jest *pozytywne* w φ).

Zwróćmy uwagę, że jedna zmienna może występować w formule zarówno pozytywnie, jak i negatywnie: najprostszym przykładem jest formuła $p \Rightarrow p$, w której p występuje tak pozytywnie (po prawej stronie implikacji), jak i negatywnie (po jej lewej stronie). Intuicyjnie, możemy powiedzieć że wystąpienie zmiennych jest pozytywne dokładnie wtedy gdy znajduje się po lewej stronie parzystej liczby implikacji.

Powyższe pojęcie parzystości możemy łatwo rozszerzyć na dowody w notacji z poprzedniego zadania: zmienne w dowodzie zachowują swój znak, chyba że znajdują się w formule wprowadzonej przez ramkę (po lewej stronie dwukropka) — w tym ostatnim przypadku zmieniają swój znak.

Zaimplementuj funkcje znajdujące zbiory zmiennych występujących na pozytywnych/negatywnych pozycjach w danej formule/dowodzie i przetestuj je na przykładowych danych.