Programowanie funkcyjne 2016

Grupa kzi
Lista 1
(11.10.2016)

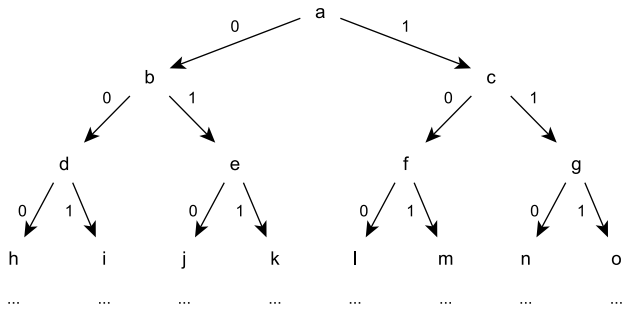
Wstęp. W poprzedniej edycji tej pracowni na Liście 1 studenci mieli zauważyć, że skoro λ-rachunek jest Turing zupełny, to wypadałoby, żeby za pomocą λ dawało się zdefiniować listy (dla uproszczenia dopuściliśmy jeszcze użycie liczb całkowitych i wyjątków). W tym roku spróbujemy w ten sposób zdefiniować drzewa binarne.

W przypadku list definiowaliśmy je za pomocą funkcji (częściowych) z pozycji w wartości. Np. funkcja f: `int -> int` taka, że

$$f\ i = \begin{cases} 5, & i = 0 \\ 10, & i = 1 \\ 15, & i = 2 \\ \text{rzuca wyjątek}, & \text{w.p.p.} \end{cases}$$

reprezentowała listę $[5, 10, 15]$. W przypadku drzew binarnych można zrobić podobnie, tylko pozycje są nieco bardzie skomplikowane.

Spójrzmy na kawałek binarnego drzewa poniżej. Jak widać pozycję każdego wierzchołka można opisać jako ciąg zer i jedynek, gdzie pusty ciąg oznacza korzeń, a każde 0 bądź 1 reprezentuje, odpowiednio, skręt w lewo bądź skręt w prawo. Stąd blisko już do zapisania pozycji za pomocą liczb naturalnych. Problemem jest korzeń, bo nie mamy liczb 0-cyfrowych, i zera na końcu ciągu odpowiadającemu najbardziej znaczącym bitom liczby. Ale po kursie Logiki powinni Państwo umieć sobie radzić z takimi problemami (chyba).



Uwaga†: Jeśli zadanie jest niedospecyfikowane, to braki w specyfikacji można uzupełnić w dowolny sensowny(!) sposób.

Zadania w OCamlu

Na potrzeby tej listy zakładamy, że `int` w OCamlu jest typem wszystkich liczb całkowitych, tak jak `Integer` w Haskellu. Przyjmujemy też standardową konwencję mówiącą, że funkcje wywołane na niepoprawnych danych mogą robić cokolwiek.

1. Korzystając jedynie z typu `int`, zmiennych typowych i typu funkcyjnego zdefiniuj typ `'a flbt` (Forgetful Lazy Binary Tree), którego wartości będą reprezentowały skończone i nieskończone drzewa binarne przechowujące wartości typu `'a`, zgodnie z naszkicowaną powyżej konwencją. Należy ponadto przyjąć jedno z poniższych założeń i zaznaczyć swój wybór w komentarzu.

   a) Jeśli pozycja nie występuje w drzewie, to funkcja reprezentująca to drzewo w typie `'a flbt` rzuca dla tej pozycji jakiś wyjątek. Warto zwrócić uwagę, że przy tym założeniu funkcja, np., zwracająca `1` dla pozycji korzenia, `2` dla pozycji lewego dziecka lewego dziecka korzenia, i rzucająca wyjątek dla wszystkich innych pozycji nie jest poprawnym drzewem.

   b) Jeśli pozycja nie występuje w drzewie, to funkcja reprezentująca to drzewo w typie `'a flbt` rzuci dla niej, bądź dla któregoś z jej przodków jakiś wyjątek. W tym przypadku powyższa funkcja reprezentuje 1-elementowe drzewo przechowujące w korzeniu wartość `1`.

   Prosta obsługa wyjątków jest opisana tutaj: moja strona domowa → Programowanie funkcyjne 2016 → Dodatkowe informacje. (1 pkt)

2. Zdefiniuj nieskończone drzewo `char`-ów, którego nieskończone ścieżki zaczynające się w korzeniu zawierają wszystkie napisy postaci $\texttt{a}\,(\texttt{a}|\texttt{b})^{\omega}$, tj. nieskończone napisy składające się z liter `a`, `b` i zaczynające się na `a`. (3 pkt)

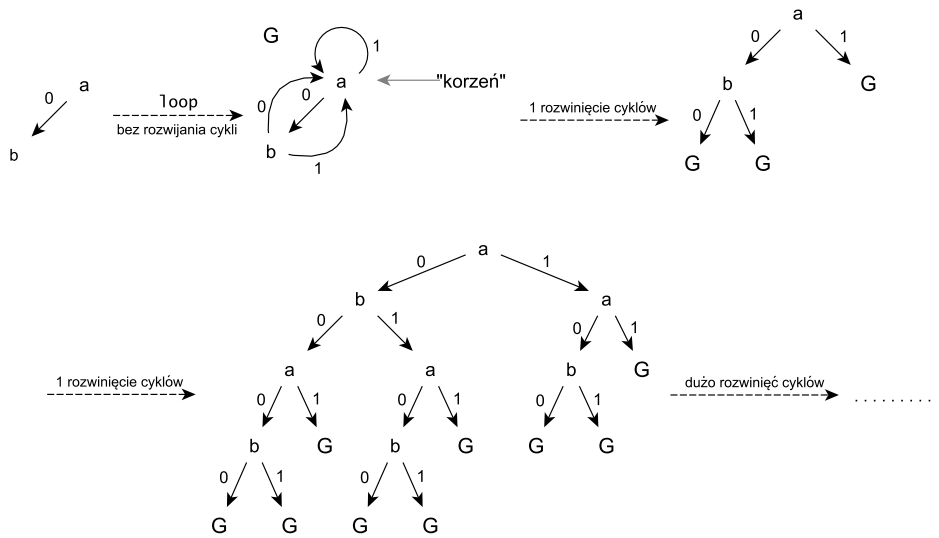
3. Zdefiniuj

   - stałą `pos_root` przechowującą pozycję korzenia,
   - funkcje `pos_left_child` i `pos_right_child`, które zaaplikowane do pozycji zwracają odpowiednio pozycję dziecka na lewo i prawo od niej. (2 pkt)

4. Zdefiniuj

   - funkcję `is_empty`, która zaaplikowana do drzewa zwraca `true` jeśli jest ono puste i `false` w.p.p.,
   - funkcję `get`, która zaaplikowana do drzewa i pozycji zwraca wartość przechowywaną na tej pozycji w tym drzewie, a jeśli taka nie istnieje rzuca jakiś wyjątek. (1 pkt w przypadku wybrania warunku a) w Zadaniu 1 lub 3 pkt w przypadku wybrania b))

5. Zdefiniuj
   - stałą `empty` przechowującą drzewo puste typu `'a flbt`,
   - funkcję `join`, która zaaplikowana do drzew s, t o elementach tego samego typu i wartości v, także tego typu, zwraca drzewo, gdzie
     - w korzeniu przechowywane jest v, oraz
     - jego bezpośrednimi poddrzewami są s, t.
   - funkcje `left_subtree` i `right_subtree`, które zaaplikowane do niepustego drzewa zwracają odpowiednio jego lewe i prawe bezpośrednie poddrzewo. (3 PKT)
6. Zdefiniuj funkcję `depth`, która oblicza głębokość skończonego drzewa. (Czy da się napisać `depth`a, który działała także dla drzew nieskończonych można się dowiedzieć na JFiZO.) (2 PKT)
7. Zdefiniuj funkcję `memorize`, która tworzy kopię skończonego drzewa taką, że pobranie wartości należącego do niego wierzchołka ma złożoność $\mathcal{O}$ od głębokości tego wierzchołka. W zadaniu nie można korzystać z dodatkowych struktur danych. (5 PKT)
8. Zdefiniuj funkcję `loop`, która zaaplikowana do drzewa t zwraca t, w którym w miejsce wszystkich brakujących dzieci zostało podstawione t, a następnie w miejsce wszystkich brakujących dzieci, które się pojawiły znów zostało podstawione t, i tak w nieskończoność. Równoważnie można powiedzieć, że wynikiem ma być drzewo, które powstaje przez wstawienie w miejsce wszystkich brakujących krawędzi w t krawędzi prowadzących do korzenia t i rozwinięcie powstałych cyklów w nieskończoność. Co to znaczy ilustruje poniższy obrazek.



(5 PKT)