MP20 @ II UWr 31 marca 2020 r.

Lista zagadnień nr 5

Zamiast zajęć

W tym tygodniu podsumowujemy pierwszą część wykładu. Będziemy pracować z bardziej skomplikowanymi strukturami danych, formułować ich własności i dowodzić o nich twierdzenia; zastanowimy się też nad problemem alokacji pamięci, żeby móc lepiej określić jak sprawnie działają nasze programy. Przed zajęciami należy przeczytać **Rozdział 2.4** podręcznika, potrafić **sformułować i użyć twierdzenia o indukcji** dla drzewiastych struktur danych, a także rozumieć pojęcie **nieużytku** i ich wpływu na działanie i złożoność programów.

Formuły rachunku zdań

W tym tygodniu będziemy zajmować się przekształcaniem formuł rachunku zdań i różnymi ich własnościami znanymi z logiki, z poprzedniego semestru. Zaczynamy od definicji predykatu `prop?` opisującego formuły rachunku zdań i predykatów ustalających reprezentację poszczególnych konstrukcji.

```
(define (var? t)
  (symbol? t))

(define (neg? t)
  (and (list? t)
       (= 2 (length t))
       (eq? 'neg (car t))))

(define (conj? t)
  (and (list? t)
       (= 3 (length t))
       (eq? 'conj (car t))))

(define (disj? t)
  (and (list? t)
       (= 3 (length t))
       (eq? 'disj (car t))))
```

```
(define (prop? f)
  (or (var? f)
      (and (neg? f)
           (prop? (neg-subf f)))
      (and (disj? f)
           (prop? (disj-left f))
           (prop? (disj-rght f)))
      (and (conj? f)
           (prop? (conj-left f))
           (prop? (conj-rght f)))))
```

Ćwiczenie 1.

Zdefiniuj konstruktory `neg`, `conj` i `disj`, a także odpowiednie selektory odpowiadające powyższym predykatom.

Ćwiczenie 2.

Sformułuj zasadę indukcji strukturalnej dla formuł rachunku zdań wynikającą z predykatu `prop`?.

Ćwiczenie 3.

Zdefiniuj procedurę `free-vars` znajdującą zbiór *zmiennych wolnych* formuły, reprezentowany za pomocą listy. Ponieważ w formułach zdaniowych zmienne nie są wiązane, wystarczy znaleźć zbiór wszystkich zmiennych występujących w formule.

Ćwiczenie 4.

Poniższa procedura definiuje zbiór wszystkich wartościowań dla danego zbioru zmiennych. Zdefiniuj procedurę `eval-formula` przyjmującą formułę i wartościowanie, i obliczającą wartość logiczną formuły dla tego wartościowania. Jeśli w formule występują zmienne nie zdefiniowane w wartościowaniu, zgłoś błąd.

```
(define (gen-vals xs)
  (if (null? xs)
      (list null)
      (let*
          ((vss  (gen-vals (cdr xs)))
           (x    (car xs))
           (vst  (map (lambda (vs) (cons (list x true)  vs)) vss))
           (vsf  (map (lambda (vs) (cons (list x false) vs)) vss)))
        (append vst vsf))))
```

Następnie przy użyciu wcześniej zdefiniowanych procedur zdefiniuj procedurę `falsifiable-eval?` przyjmującą formułę logiczną, i zwracającą wartościowanie przy którym jest ona fałszywa lub `false` jeśli jest ona tautologią.

Ćwiczenie 5.

Zdefiniuj predykat `nnf?` który zachodzi gdy jego argument jest formułą zdaniową w negacyjnej postaci normalnej. Żeby wygodnie przedstawić takie formuły najlepiej zdefiniować dodatkowy typ danych reprezentujący literały (tj. potencjalnie zanegowane zmienne zdaniowe). Rozszerz procedury `free-vars`

i `eval-formula` tak żeby działały zarówno dla zwykłych formuł zdaniowych, jak i dla formuł w postaci negacyjnej.

Ćwiczenie 6.

Zdefiniuj procedurę `convert-to-nnf` przekształcającą formułę do równoważnej jej formuły w negacyjnej postaci normalnej. Zadbaj o to żeby translacja była *strukturalna*, tj. żeby rekurencyjnie wywoływać ją wyłącznie na podformułach argumentu. **Wskazówka:** może przydać Ci się dodatkowa procedura, która będzie *wzajemenie rekurencyjna* z procedurą `convert-to-nnf`.

Ćwiczenie 7.

Udowodnij że `convert-to-nnf` transformuje formuły (czyli dane spełniające `prop?`) do negacyjnej postaci normalnej (czyli na dane spełniające `nnf?`). Następnie udowodnij, że transformacja ta zachowuje wynik wartościowania: dla dowolnych f i v, jeśli `(prop?` f`)`, to

`(eval-formula` f v`)` $\equiv$ `(eval-formula (convert-to-nnf` f`)` v`)`

Ćwiczenie 8.

Formuły w *koniunkcyjnej postaci normalnej* mają znacznie prostszą strukturę niż formuły w NNFie: możemy zareprezentować je jako *listy* koniunktów, z których każdy jest listą literałów. Przyjmując że literał zdefiniowaliśmy wcześniej (w zadaniu o NNFie), możemy zatem zdefiniować formuły w CNF następująco:

```
(define (clause? x)
  (and (list? x)
       (andmap lit? x)))

(define (cnf? x)
  (and (list? x)
       (andmap clause? x)))
```

Zdefiniuj procedurę `convert-to-cnf` przekształcającą formułę w NNFie do CNFu. Tak jak poprzednio translacja powinna być strukturalna, trudność polega na zrozumieniu co zrobić z formułami w CNFie otrzymanymi z translacji podformuł. Następnie zdefiniuj procedurę `eval-cnf` obliczającą wartość formuły w CNFie. Czy potrafisz udowodnić że `convert-to-cnf` zachowuje wartość formuły przy dowolnym wartościowaniu?

Zadania domowe

Ćwiczenie 9.

Zdefiniuj procedurę `falsifiable-cnf?` która przyjmuje formułę zdaniową, tłumaczy ją do CNFu i na tej podstawie znajduje fragment wartościowania który falsyfikuje wejściową formułę (lub zwraca `false` gdy formuła jest tautologią). Twoja procedura nie powinna generować wszystkich wartościowań, a tylko opierać się na strukturze otrzymanej formuły w CNFie.

Porównaj działanie dwóch procedur sprawdzających czy formuła jest tautologią. Czy pojawiają się różnice w efektywności? Opisz obserwacje i swoje wnioski w komentarzu w pliku z rozwiązaniem.

Uwaga: Dodatkowe zadania domowe mogą pojawić się w późniejszym terminie (będziemy informować).