Metody programowania 2017

Lista zadań na pracownię nr 1

Na tej liście zadań zajmiemy się problemem spełnialności zbioru klauzul. Przypomnijmy, że klauzula to alternatywa skończenie wielu literałów, a literał to zmienna zdaniowa lub jej negacja. Aby w Prologu zapisywać klauzule w czytelny dla nas sposób uczyńmy `~/1` operatorem prefiksowym, a `v/2` operatorem infiksowym oraz zdefiniujmy ich priorytet i łączność:

```
:- op(200, fx, ~).
:- op(500, xfy, v).
```

Przyjmijmy również następujące definicje:

zmienna zdaniowa: dowolny atom różny od `[]`,

literał: term postaci x lub `~` x, gdzie x jest zmienną zdaniową

klauzula pusta: atom `[]`,

klauzula niepusta: literał lub term postaci l `v` c, gdzie l jest literałem, a c jest klauzulą niepusta,

klauzula: klauzula pusta lub klauzula niepusta.

Poprawnymi klauzulami są na przykład `p v ~p`, `[]` oraz `~q v p v p`. Zbiory klauzul będziemy reprezentować za pomocą list.

Na kursie *Logiki dla informatyków* zdefiniowaliśmy wartościowanie jako funkcję ze zbioru zmiennych w zbiór zawierający dwie stałe reprezentujące prawdę i fałsz. Z punktu widzenia spełnialności (skończonego) zbioru klauzul interesuje nas tylko wartość tej funkcji w skończenie wielu punktach, tzn. dla zmiennych występujących w danym zbiorze klauzul, dlatego wartościowania będziemy reprezentować za pomocą list. Przyjmijmy następujące definicje:

wartość logiczna: atom `t` lub atom `f`,

wartościowanie zmiennej: term postaci `(`x`,`v`)`, gdzie x jest zmienną zdaniową, a v jest wartością logiczną,

wartościowanie: lista wartościowań zmiennych.

Dodatkowo wymagamy, że wartościowanie dla zbioru klauzul C definiuje wartości tylko tych zmiennych, które występują w C oraz by każda zmienna występująca w C występowała w wartościowaniu dokładnie raz.

Na przykład dla zbioru klauzul `[p v ~p, q]` możliwe wartościowania to między innymi `[(p,t),(q,f)]` oraz `[(q,t),(p,t)]`, natomiast `[(p,t),(q,t),(p,t)]`, `[(q,t)]` i `[(p,t),(q,t),(r,f)]` nie są poprawnymi wartościowaniami.

Czasami, żeby przekonać się o spełnialności danego zbioru klauzul, wystarczy podać wartościowanie tylko niektórych zmiennych, natomiast wartościowanie pozostałych może być dowolne. Np. aby zbiór `[p v ~p, q]` był spełniony, wystarczy zmiennej `q` przypisać wartość `t`. Dlatego chcąc w zwięzły sposób zapisywać zbiory wartościowań spełniających dany zbiór klauzul, wprowadzimy nową wartość logiczną `x` oznaczającą „dowolną wartość”. A dokładniej mówiąc, wprowadźmy następujące definicje:

uogólniona wartość logiczna: atom `t`, atom `f` lub atom `x`,

uogólnione wartościowanie zmiennej: term postaci (x, v), gdzie x jest zmienną zdaniową, a v jest uogólnioną wartością logiczną,

uogólnione wartościowanie: lista uogónionych wartościowań zmiennych.

Dodatkowo na uogólnione wartościowania nakładamy wymagania dotyczące unikatowości zmiennych analogiczne do wymagań nakładanych na zwykłe wartościowania.

Powiemy, że wartościowanie σ jest **instancją** uogólnionego wartościowania ρ, jeśli zgadzają się one na wszystkich zmiennych, dla których ρ nie przypisuje wartości `x`. Uogólnione wartościowanie ρ **spełnia** zbiór klauzul C, jeśli C jest spełnione przez wszystkie instancje ρ.

Zadanie, część 1.
Termin zgłaszania w serwisie SKOS: 20 marca 2017 6:00 AM CET

Napisz zestaw testów dla problemu szukania wartościowania spełniającego zbiór klauzul. Należy posłużyć się następującym szablonem (znajdującym się również w serwisie SKOS):

```
% Definiujemy moduł zawierający testy.
% Należy zmienić nazwę modułu na {imie}_{nazwisko}_tests gdzie za
% {imie} i {nazwisko} należy podstawić odpowiednio swoje imię
% i nazwisko bez znaków diakrytycznych
:- module(imie_nazwisko_tests, [tests/5]).

% definiujemy operatory ~/1 oraz v/2
:- op(200, fx, ~).
:- op(500, xfy, v).

% Zbiór faktów definiujących testy
% Należy zdefiniować swoje testy
tests(excluded_middle, validity, [p v ~p], 500, solution([(p,t)])).
```

rozszerzając definicję predykatu `tests(-Name,-Type,-Input,-Timeout,-Ans)` o nowe fakty. Oto znaczenia poszczególnych parametrów:

`Name`: atom reprezentujący nazwę testu. Nazwa powinna mówić coś o teście i jednoznacznie go identyfikować.

`Type`: atom reprezentujący typ testu. Powinien przyjąć jedną z następujących wartości:

- `validity` oznacza test poprawnościowy. Testy poprawnościowe powinny być małe, tak by naiwne, ale poprawne rozwiązanie bez problemu mieściło się w limitach czasowych. Za tą grupę testów można zdobyć najwięcej punktów, dlatego postaraj się, by pokryła ona jak najwięcej brzegowych przypadków.
- `performance` oznacza test wydajnościowy. Limity czasowe dla testów wydajnościowych powinny odróżniać powolne rozwiązania od tych szybkich.

`Input`: zbiór klauzul będący danymi wejściowymi.

`Timeout`: liczba całkowita z przedziału 500–10000 oznaczająca limit czasowy dla tego testu wyrażony w milisekundach.

`Ans`: term reprezentujący oczekiwaną odpowiedź. Powinien mieć jedną z następujących postaci:

- `solution(`σ`)` gdzie σ jest jest wartościowaniem spełniającym `Input`;
- `count(`n`)` gdzie n jest liczbą różnych wartościowań (modulo kolejność zmiennych) spełniających `Input`

Wymogi formalne
Należy zgłosić pojedynczy plik o nazwie *imię_nazwisko*`_tests.pl` gdzie za *imię* i *nazwisko* należy podstawić odpowiednio swoje imię i nazwisko bez znaków diakrytycznych. Nadesłany plik powinien być kodem źródłowym napisanym w Prologu, który definiuje moduł eksportujący jeden predykat `tests/5` tak jak to opisano w załączonym szablonie. **Rozwiązania nie spełniające wymogów formalnych nie będą oceniane!**

Zadanie, część 2.
Termin zgłaszania w serwisie SKOS: 27 marca 2017 6:00 AM CEST

Napisz program szukający wartościowań spełniających zadany zbiór klauzul. Należy posłużyć się następującym szablonem (znajdującym się również w serwisie SKOS):

```
% Definiujemy moduł zawierający rozwiązanie.
% Należy zmienić nazwę modułu na {imie}_{nazwisko} gdzie za
% {imie} i {nazwisko} należy podstawić odpowiednio swoje imię
% i nazwisko bez znaków diakrytycznych
:- module(imie_nazwisko, [solve/2]).

% definiujemy operatory ~/1 oraz v/2
:- op(200, fx, ~).
:- op(500, xfy, v).

% Główny predykat rozwiązujący zadanie.
% UWAGA: to nie jest jeszcze rozwiązanie; należy zmienić jego
% definicję.
solve(Clauses, Solution) :-
  Clauses  = [p v ~p],
  Solution = [(p,x)].
```

zmieniając definicję predykatu `solve(+Clauses, -Solution)`. Predykat ten powinien dla danego zbioru klauzul `Clauses` z każdym nawrotem generować uogólnione wartościowania spełniające `Clauses`. Cała przestrzeń rozwiązań powinna być pokryta, tzn. każde wartościowanie spełniające `Clauses` powinno być instancją **dokładnie jednego** uogólnionego wartościowania `Solution` generowanego przez wywołanie `solve(Clauses, Solution)`. Możesz założyć, że dane wejściowe są poprawne.

Wymogi formalne
Należy zgłosić pojedynczy plik o nazwie *imię_nazwisko*`.pl` gdzie za *imię* i *nazwisko* należy podstawić odpowiednio swoje imię i nazwisko bez znaków diakrytycznych. Nadesłany plik powinien być kodem źródłowym napisanym w Prologu, który definiuje moduł eksportujący jeden predykat `solve/2` tak jak to opisano w załączonym szablonie. **Rozwiązania nie spełniające wymogów formalnych nie będą oceniane!**

Uwaga
W serwisie SKOS umieszczono plik `prac1.pl` pozwalający uruchamiać napisane rozwiązanie na przygotowanych testach. Sposób jego uruchamiania znajduje się w komentarzu wewnątrz pliku.