Metody programowania 2017

Lista zadań na pracownię nr 2

Na poprzedniej liście zadań badaliśmy, czy zadany zbiór klauzul jest spełnialny. Robiliśmy to poprzez szukanie wartościowania który go spełnia. Z kursu *Logiki dla informatyków* wiemy, że ten problem można rozwiązać w zupełnie inny sposób: za pomocą rezolucji. Przypomnijmy kilka definicji oraz prostych faktów.

Niech C będzie klauzulą nie zawierającą literału p, a D będzie klauzulą nie zawierającą literału $\sim p$. Wówczas **rezolwentą** klauzul $C \cup \{p\}$ oraz $D \cup \{\sim p\}$ (reprezentowanych jako zbiór literałów) względem zmiennej p nazwiemy klauzulę postaci $C \cup D$. Zauważmy, że każde wartościowanie spełniające klauzule $C \cup \{p\}$ oraz $D \cup \{\sim p\}$ spełnia również ich rezolwentę, więc rozszerzenie zbioru klauzul o jego rezolwenty nie zmienia jego niesprzeczności. A zatem, jeśli ze zbioru klauzul A wyprowadzimy klauzulę pustą za pomocą rezolucji, to zbiór A jest sprzeczny. Zachodzi również twierdzenie odwrotne: jeśli zbiór klauzul jest sprzeczny, to da się wyprowadzić z niego klauzulę pustą.

Wprowadźmy następujące pojęcie, które uchwyci intuicję dotyczącą wyprowadzania klauzuli pustej. **Rezolucyjnym dowodem sprzeczności** zbioru klauzul A nazwiemy ciąg klauzul zakończony klauzulą pustą, taki, że każdy element tego ciągu jest albo elementem zbioru A, albo jest rezolwentą dwóch klauzul występujących wcześniej w tym ciągu. Wówczas zbiór klauzul jest spełnialny, wtedy i tylko wtedy, gdy nie ma rezolucyjnego dowodu sprzeczności.

Zadanie polega na zaprogramowaniu w Prologu dwóch predykatów: `resolve/4` oraz `prove/2` które odpowiednio wyliczają rezolwentę dwóch klauzul oraz szukają rezolucyjnego dowodu sprzeczności. Będziemy używać formatu klauzul zdefiniowanego na poprzedniej liście zadań.

Zadanie, część 1.
Termin zgłaszania w serwisie SKOS: 3 kwietnia 2017 6:00 AM CEST

Napisz zestaw testów dla obliczania rezolwenty dwóch klauzul oraz szukania rezolucyjnego dowodu sprzeczności. 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 wielkich liter oraz znaków diakrytycznych
:- module(imie_nazwisko_tests, [resolve_tests/5, prove_tests/4]).

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

% Zbiór faktów definiujących testy dla predykatu resolve
% Należy zdefiniować swoje testy
resolve_tests(simple_test, q, p v q, ~q v r, p v r).
```

```
% Zbiór faktów definiujących testy dla predykatu prove
% Należy zdefiniować swoje testy
prove_tests(example, validity, [p v q v ~r, ~p v q, r v q, ~q, p], unsat).
prove_tests(excluded_middle, validity, [p v ~p], sat).
```

rozszerzając definicje predykatów `resolve_tests/5` oraz `prove_tests/4`. Znaczenia poszczególnych parametrów tych predykatów znajdują się poniżej.

Predykat `resolve_tests(-Name,-Var,-Clause1,-Clause2,-Resolvent)`:

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

`Var`: atom reprezentujący zmienną, względem której należy wykonać rezolucję.

`Clause1`: pierwsza klauzula do rezolucji. Powinna zawierać pozytywne wystąpienie zmiennej `Var` (literał `Var`).

`Clause2`: pierwsza klauzula do rezolucji. Powinna zawierać negatywne wystąpienie zmiennej `Var` (literał `~Var`).

`Resolvent`: Rezolwenta klauzul `Clause1` oraz `Clause2` względem zmiennej `Var`.

Predykat `prove_tests(-Name,-Type,-Clauses,-Ans)`:

`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 naiwny, ale poprawny program bez problemu znalazł rozwiązanie w ułamek sekundy. Za tą grupę testów można zdobyć najwięcej punktów, dlatego postaraj się, by pokryła ona jak najwięcej brzegowych przypadków.
- `performance` oznacza test wydajnościowy rozróżniający programy z różnymi usprawnieniami od tych bardziej naiwnych. Program implementujący pewne usprawnienie powinien rozwiązać taki test w mniej niż kilka sekund, w przeciwieństwie do gorszych programów, które na takim teście powinny działać długo.

`Clauses` lista klauzul będąca danymi wejściowymi dla predykatu `prove`.

`Ans` atom reprezentujący spodziewany wynik dla tego testu. Powinien przyjąć jedną z wartości `unsat` albo `sat` oznaczających odpowiednio istnienie i nieistnienie rezolucyjnego dowodu sprzeczności (niespełnialność i spełnialność).

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 wielkich liter oraz polskich znaków diakrytycznych. Nadesłany plik powinien być kodem źródłowym napisanym w Prologu, który definiuje moduł eksportujący tylko predykaty

`resolve_tests/5` oraz `prove_tests/4` tak jak to opisano w załączonym szablonie. **Rozwiązania nie spełniające wymogów formalnych nie będą oceniane!**

Zadanie, część 2.
Termin zgłaszania w serwisie SKOS: 10 kwietnia 2017 6:00 AM CEST

Napisz moduł eksportujący predykat `resolve/4` obliczający rezolwentę dwóch klauzul oraz predykat `prove/2` szukający rezolucyjnego dowodu sprzeczności. 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 wielkich liter oraz znaków diakrytycznych
:- module(imie_nazwisko, [resolve/4, prove/2]).

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

% Szukanie rezolwenty.
% UWAGA: to nie jest jeszcze rozwiązanie; należy zmienić definicję
% tego predykatu
resolve(q, p v q, ~q v r, r v p).

% Główny predykat rozwiązujący zadanie.
% UWAGA: to nie jest jeszcze rozwiązanie; należy zmienić jego
% definicję.
prove(Clauses, Proof) :-
  Clauses = [p v q v ~r, ~p v q, r v q, ~q, p],
  Proof   = [(p v q v ~r, axiom), (~p v q, axiom), (q v ~r, (p, 1, 2)),
    (r v q, axiom), (q, (r, 4, 3)), (~q, axiom), ([], (q, 5, 6))].
```

Predykat `resolve(+Var, +Clause1, +Clause2, -Resolvent)` powinien unifikować zmienną `Resolvent` z klauzulą będącą rezolwentą klauzul `Clause1` oraz `Clause2` względem zmiennej `Var`. Możesz założyć, że zmienna `Var` ma pozytywne wystąpienie w klauzuli `Clause1` oraz negatywne wystąpienie w klauzuli `Clause2`. Dla poprawnych danych wejściowych wywołanie tego predykatu powinno zakończyć się pojedynczym sukcesem.

Predykat `prove(+Clauses, -Proof)` powinien dla zadanego zbioru klauzul `Clauses` (reprezentowanego za pomocą listy) wyprodukować rezolucyjny dowód sprzeczności `Proof`, lub zawieść jeśli taki dowód nie istnieje. Dowód `Proof` powinien być listą par postaci `(Clause, Origin)` gdzie `Clause` jest klauzulą, a `Origin` termem opisującym jej pochodzenie. Możliwe wartości pochodzenia `Origin` to

`axiom`: ten atom oznacza, że klauzula `Clause` jest jedną z klauzul wejściowych `Clauses`.

`(Var, N, M)`: taka trójka oznacza, że klauzula `Clause` jest rezolwentą klauzul występujących w dowodzie na pozycjach `N` oraz `M` (licząc od 1) względem zmien-

nej `Var`. Oczywiście bieżąca klauzula `Clause` powinna znajdować się na pozycji większej niż `N` oraz `M`. Dodatkowo wymagamy, by klauzula na pozycji `N` zawierała pozytywne wystąpienie zmiennej `Var`, a klauzul na pozycji `M` zawierała jej negatywne wystąpienie.

Dowód musi kończyć się klauzulą pustą. Postaraj się, by nie zawierał nadmiarowych klauzul, tzn. takich, które nigdzie nie są wykorzystywane do rezolucji. W zadaniu nie specyfikujemy, co powinien zrobić predykat `prove` przy kolejnych nawrotach. W szczególności może znaleźć inny dowód, lub po prostu zawieść. Poniżej przedstawiamy przykładowe wywołanie predykatu `prove`.

```
?- prove([p v q v ~r, ~p v q, r v q, ~q, p], Proof).
Proof = [(p v q v ~r, axiom), (~p v q, axiom), (q v ~r, (p, 1, 2)),
  (r v q, axiom), (q, (r, 4, 3)), (~q, axiom), ([], (q, 5, 6))].
```

Wymogi formalne
Należy zgłosić pojedynczy plik o nazwie *imię_nazwisko*`.pl` gdzie za *imię* i *nazwisko* należy podstawić odpowiednio swoje imię i nazwisko bez wielkich liter oraz znaków diakrytycznych. Nadesłany plik powinien być kodem źródłowym napisanym w Prologu, który definiuje moduł eksportujący tylko predykaty `resolve/4` oraz `prove/2` tak jak to opisano w załączonym szablonie. **Rozwiązania nie spełniające wymogów formalnych nie będą oceniane!**

Uwaga
W serwisie SKOS umieszczono plik `prac2.pl` pozwalający uruchamiać napisane rozwiązanie na przygotowanych testach. Sposób jego uruchamiania znajduje się w komentarzu wewnątrz pliku.