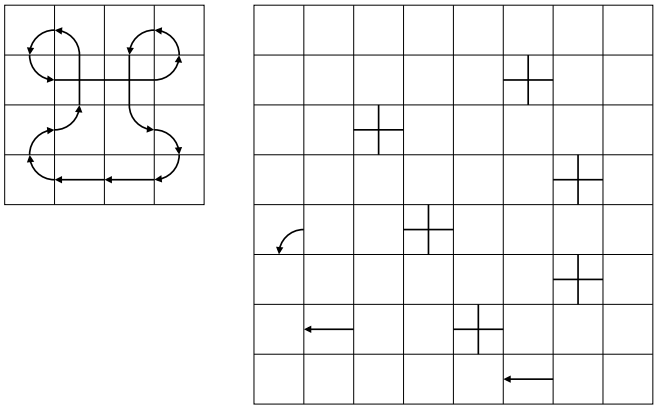
Kurs języka Prolog 2019

Lista zadań nr 5

Na zajęcia 3 kwietnia 2019

Zadanie 1 (8 pkt). Napisz program rozwiązujący łamigłówkę *Autodrom* z numeru 7/2001 miesięcznika „Wiedza i Życie": *Oznacz na diagramie tor wyścigowy, którego niektóre fragmenty,* [w] *tym wszystkie skrzyżowania (oczywiście bezkolizyjne)* [—] *ujawniono. Trasa jest pętlą przechodzącą przez wszystkie pola diagramu*[,] *przez każde (poza skrzyżowaniami) tylko raz. Dla ułatwienia przy niektórych ujawnionych fragmentach oznaczony jest kierunek jazdy obowiązujący na całym torze, a* [poniżej (w oryginale *powyżej*)] *zamieszczony jest przykład.*



Jeśli łamigłówka nie przypadła Ci do gustu, możesz wybrać z „Puzelandu" inną, dostatecznie ciekawą (np. *Koraliki* (10/2001), *Magnesy* (12/2001), *Na kempingu* (11/2000), *Wypustki* (8/2000) itp.).

Zadanie 2 (6 pkt). Rozważmy struktury zbudowane z liczb, atomów i funktorów `(+)/2`, `(-)/2`, `(*)/2`, `(/)/2`, `sqrt/1`, `sin/1`, `cos/1`, `tan/1`, `ctg/1`, `log/1`, `log10/1`, `exp/1`, `(**)/2`, `e/0`, `pi/0`. Dowolny atom różny od `e/0` i `pi/0` jest traktowany jako nazwa zmiennej. Zaprogramuj predykat `vars/2` ujawniający nazwy zmiennych występujących w wyrażeniu. Zaprogramuj predykat `eval/3` który dla podanego wyrażenia i listy asocjacji przypisującej zmiennym wartości liczbowe wyznacza wartość wyrażenia, np. obliczenie poniższego celu

```
eval(2 * x + 3 * y, [(x,5),(y,7)], X)
```

unifikuje zmienną `X` z liczbą 31. Zaprogramuj predykat `diff/3` wyznaczający pochodną podanego wyrażenia względem podanej zmiennej, np. obliczenie celu

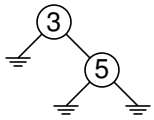
```
diff(x**3, x, X)
```

powoduje zunifikowanie zmiennej `X` ze strukturą `3 * x**2`. Czy uruchomienie predykatu `diff` w trybie `(-,+,+)` pozwala na wyznaczanie całek nieoznaczonych? Zaprogramuj predykat `simplify/2` upraszczający podane wyrażenie, np. obliczenie celu

```
simplify(2 * x + 1 * (y + 0) - 2 * x + 0 * z, X)
```

podstawia pod zmienną `X` strukturę `y`. *Uwaga:* dla funkcji wymiernych istnieje postać kanoniczna, zatem równość funkcji wymiernych jest rozstrzygalna. Jeśli dodamy choć jedną funkcję oraz stałą przestępną, wówczas równość wyrażeń staje się nierozstrzygalna. Program upraszczający wyrażenia może być więc oparty jedynie na heurystykach.

Zadanie 3 (3 pkt). Drzewa binarne o etykietowanych wierzchołkach wewnętrznych zapisujemy w Prologu używając funktora `node/3` do reprezentowania wierzchołków wewnętrznych i atomu `leaf/0` do reprezentowania liści. Dla przykładu struktura `node(leaf, 3, node(leaf, 5, leaf))` reprezentuje drzewo



Zaprogramuj predykat `insert/3` operujący na drzewach BST (względem standardowego porządku `(@=<)/2`) w opisanej wyżej reprezentacji. Obliczenie celu `insert(E,T,R)` ma różny skutek zależnie od typów argumentów. Parametr `E` może być dowolną strukturą. Ponieważ w drzewie można przechowywać dowolne termy prologowe, parametr `E` jest parametrem wejściowym nawet wówczas, gdy jest nieukonkretnioną zmienną. Parametry `T` i `R` powinny być nieukonkretnionymi zmiennymi bądź atomem `leaf/0` lub strukturą zbudowaną za pomocą funktora `node/3`. Jeśli parametry `T` i `R` nie są powyższej postaci, to należy zgłosić wyjątek `domain_error`.

- Jeśli parametr `T` nie jest wolną zmienną zmienną, to powinien być atomem `leaf/0` lub strukturą zbudowaną z funktora `node/3`. W przeciwnym razie należy zgłosić wyjątek `domain_error`. Parametr `R` może być dowolną strukturą. Cel `insert(E,T,R)` wstawia element `E` do drzewa `T` i unifikuje wynikowe drzewo z `R`.
- Jeśli `T` jest wolną zmienną, to `R` powinien być atomem `leaf/0` lub strukturą zbudowaną z funktora `node/3`. W przeciwnym razie należy zgłosić wyjątek `domain_error`. Cel `insert(E,T,R)` usuwa element `E` z drzewa `R` i unifikuje wynikowe drzewo z `T`.

Zaprogramuj także predykaty `minel/3` i `maxel/3` ujawniające i usuwające z drzewa odpowiednio najmniejszy i największy element, `mirror/2` — tworzący lustrzane odbicie drzewa oraz `flatten/2` — tworzący listę etykiet drzewa w porządku infiksowym.

Zadanie 4 (3 pkt). *Kolekcje* to struktury danych służące do przechowywania elementów. Niech interfejs kolekcji składa się z następujących nazw predykatów:

- `put(+E,+S,-R)` wstawia element `E` do kolekcji `S` i zwraca nową kolekcję `R`;
- `get(+S,-E,-R)` usuwa element z kolekcji `S`, podstawia go pod `E` i zwraca nową kolekcję `R`;
- `empty(?S)` sprawdza lub tworzy pustą kolekcję;
- `addall(-E, +G, +S, -R)` wstawia wszystkie wyniki podstawień pod zmienną `E`, które spełniają cel `G` (w którym zmienna `E` występuje) do kolekcji `S` i zwraca nową kolekcję `R` (ten predykat przypomina standardowe predykaty `findall/3` i `findall/4`).

Podaj dwie implementacje tego interfejsu, jedną dla stosu (użyj list zamkniętych do reprezentowania kolekcji), drugą dla kolejek FIFO (tu użyj list różnicowych).

Rozważmy skierowany graf $G = \langle V, E \rangle$. Jego wierzchołki ze zbioru V będziemy reprezentować za pomocą dowolnych struktur (przeważnie atomowych), a relację krawędzi E — za pomocą binarnego predykatu `e/2`. Dane są wierzchołki $v_1, v_2 \in V$. Ścieżkę z wierzchołka v_1 do wierzchołka v_2 w grafie G można znaleźć używając algorytmu przeszukiwania sparametryzowanego kolekcją przechowującą oczekujące wierzchołki:

1. Jeśli kolekcja przechowująca oczekujące wierzchołki jest pusta, to zakończ pracę.
2. W przeciwnym razie wyjmij wierzchołek v z kolekcji. Jeśli v nie został wcześniej odwiedzony, to odwiedź go i wstaw wszystkich jego `e`-sąsiadów do kolekcji. W przeciwnym razie odrzuć wierzchołek v. Powtórz całą procedurę.

Zaprogramuj powyższy algorytm tak, by znajdował w grafie ścieżki między podanymi wierzchołkami używając opisanego wyżej interfejsu kolekcji. Następnie użyj stosów, by otrzymać DFS oraz kolejek, by otrzymać BFS.

Przypuśćmy, że wierzchołkom grafu dodajemy *wagi* i używamy kolejek priorytetowych do przechowywania oczekujących wierzchołków. Jaki algorytm otrzymaliśmy?