Programowanie w Logice
Struktury danych
(Lista 2)

Przemysław Kobylański

Wstęp

Struktury danych wyraża się w Prologu w postaci termów, tj. symbolicznych wyrażeń.

Dotychczas poznaliśmy proste termy takie jak zmienne i stałe.

Do budowania termów złożonych służą funktory.

Nazwą funktora jest alfanumeryczny identyfikator pisany z małej litery (dokładnie tak samo jak w przypadku stałej).

Tak jak stałe interpretujemy jako konkretne obiekty z rzeczywistego albo abstrakcyjnego świata, tak funktory interpretujemy jako funkcje operujące na obiektach.

Przykładami funktorów mogą być jednoargumentowe `ojciec/1` i `matka/1`.

Funktor `ojciec` będziemy interpretować jako funkcja, która przypisuje osobie jej ojca, natomiast funktor `matka` będziemy interpretować jako funkcja, która przypisuje osobie jej matkę.

Oto przykładowe termy zbudowane z tych funktorów i ich interpretacje:

`janek`	chłopiec o imieniu Janek
`matka(janek)`	matka Janka
`ojciec(janek)`	ojciec Janka
`matka(ojciec(janek))`	babcia Janka ze strony ojca
`ojciec(matka(janek))`	dziadek Janka ze strony mamy

Załóżmy, że stałą `a` będziemy interpretować jako liczba 0 a jednoargumentowy funktor `s` jako funkcja następnika.

Dzięki nim można zdefiniować liczby naturalne i predykat `add(X, Y, Z)`, który jest prawdziwy gdy `Z` jest sumą `X` i `Y`:

```
add(a, X, X).
add(s(X), Y, s(Z)) :-
        add(X, Y, Z).
```

Taka definicja pozwala Prologowi nie tylko wyliczyć sumę ale również odejmować a nawet dzielić całkowicie przez dwa:

```
?- add(s(s(a)), s(s(a)), X).
X = s(s(s(s(a)))).

?- add(s(s(a)), X, s(s(s(a)))).
X = s(a).

?- add(X, Y, s(s(s(a)))).
X = a,
Y = s(s(s(a))) ;
X = s(a),
Y = s(s(a)) ;
X = s(s(a)),
Y = s(a) ;
X = s(s(s(a))),
Y = a ;
false.

?- add(X, X, s(s(s(s(s(s(a))))))).
X = s(s(s(a))) ;
false.
```

Szczególną rolę w Prologu odgrywa dwuargumentowy funktor kropka.

Służy on do łączenia pierwszego elementu listy (głowa listy) z listą pozostałych elementów (ogon listy). Możemy go interpretować jako funkcja, która dla danego elementu `X` i danej listy `Y`, tworzy nową listę z głową taką jak `X` i ogonem takim jak `Y`.

Listę pustą zapisuje się jako stała `[]` (dwa kwadratowe nawiasy).

Oto przykłady termów i ich interpretacji:

`[]`	lista pusta
`.(a, [])`	lista złożona z jednego elementu `a`
`.(a, .(b, []))`	lista złożona z dwóch elementów `a` i `b`
`.(X, _)`	lista o głowie `X` i nieistotnym ogonie
`.(X, .(X, _))`	lista o identycznych dwóch pierwszych elementach

Notacja list z użyciem funktora kropka jest uciążliwa. Dlatego stosuje się notację z kwadratowymi nawiasami.

W notacji tej wymienia się elementy listy między kwadratowymi nawiasami oddzielając je przecinkami.

Do oddzielenia początkowych elementów listy od listy pozostałych elementów używa się pionowej kreski:

```
[1, 2, 3, 4, 5] = [1 | [2, 3, 4, 5]]
                = [1, 2 | [3, 4, 5]]
                = [1, 2, 3 | [4, 5]]
                = [1, 2, 3, 4 | [5]]
                = [1, 2, 3, 4, 5 | []]
```

Na zakończenie wstępu podamy definicje trzech podstawowych predykatów operujących na listach:

1. `member(X, L)`, który jest prawdziwy, gdy `X` jest elementem listy `L`,
2. `append(L1, L2, L3)`, który jest prawdziwy, gdy lista `L3` jest połączeniem (konkatenacją) list `L1` i `L2`,
3. `select(X, L1, L2)`, który jest prawdziwy, gdy lista `L2` powstaje z listy `L1` przez wyjęcie jednego elementu `X`.

Oto definicje powyższych predykatów:

```
member(X, [X | _]).
member(X, [_ | L]) :-
        member(X, L).

append([], L, L).
append([X | L1], L2, [X | L3]) :-
        append(L1, L2, L3).

select(X, [X | L], L).
select(X, [Y | L1], [Y | L2]) :-
        select(X, L1, L2).
```

Zadania

Zadanie 1 (1 pkt)

Napisz predykat `środkowy(L, X)`, który jest prawdziwy jeśli `X` jest środkowym elementem listy `L`. Jeśli lista `L` ma parzystą liczbę elementów, to warunek `środkowy(L, X)` powinien zawieść.

Przykład

```
?- środkowy([1, 2, 3, 4, 5], X).
X = 3 ;
false.

?- środkowy([1, 2, 3, 4], X).
false.
```

Zadanie 2 (2 pkt)

1. Napisz predykat `jednokrotnie(X, L)`, który jest spełniony, jeśli `X` występuje dokładnie jeden raz na liście `L`.
2. Napisz predykat `dwukrotnie(X, L)`, który jest spełniony, jeśli `X` występuje dokładnie dwa razy na liście `L`.

Przykład

```
?- jednokrotnie(X, [3, 2, 4, 1, 2, 3]).
X = 4 ;
X = 1 ;
false.

?- dwukrotnie(X, [3, 2, 4, 1, 2, 3]).
X = 3 ;
X = 2 ;
false.
```

Wskazówka

W definicjach warunków `jednokrotnie/2` i `dwukrotnie/2` możesz korzystać z innych predykatów.

Zadanie 3 (3 pkt)

Dany jest graf skierowany w postaci faktów `arc(X, Y)`, wyrażających, że jest łuk od węzła `X` do węzła `Y`.

Napisz predykat `osiągalny(X, Y)`, który jest spełniony gdy węzeł `Y` jest osiągalny z węzła `X` (tzn. jest ścieżka od `X` do `Y`).

Przykład

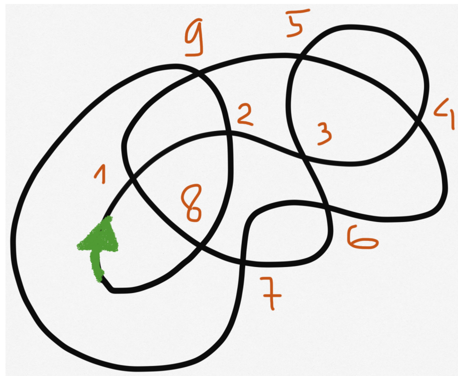
Załóżmy, że graf składa się z czterech łuków:

```
arc(a, b).
arc(b, a).
arc(b, c).
arc(c, d).
```

Wówczas:

```
?- osiągalny(a, X).
X = a ;
X = b ;
X = c ;
X = d ;
false.

?- osiągalny(b, X).
X = b ;
X = a ;
X = c ;
X = d ;
false.
```



Rysunek 1: Krzywa zamknięta z ustalonym zwrotem

```
?- osiągalny(c, X).
X = c ;
X = d ;
false.

?- osiągalny(d, X).
X = d ;
false.

?- osiągalny(X, a).
X = a ;
X = b ;
false.
```

Zadanie 4 (4 pkt)

Narysujmy na płaszczyźnie krzywą zamkniętą, przyjmijmy na niej punkt i ustalmy zwrot (jak na rysunku 1). Obchodząc krzywą zgodnie z przyjętym zwrotem, numerujmy kolejno odwiedzane przecięcia (gdy odwiedzamy je po raz pierwszy).

W przykładzie z rysunku 1, krzywa przecina się w dziewięciu punktach a kolejność odwiedzania tych punktów jest następująca:

$$1, 2, 3, 4, 5, 3, 6, 7, 8, 1, 9, 5, 4, 6, 7, 9, 2, 8.$$

Zauważ, że jeśli n jest liczbą przecięć, to ciąg taki ma $2 \cdot n$ elementów, przy czym każda liczba od 1 do n pojawia się w nim dokładnie dwa razy.

Ciekawszą własnością takiego ciągu jest to, że między dwoma dowolnymi wystąpieniami tej samej liczby, zawsze znajduje się parzysta liczba elementów. W powyższym przykładzie, między pierwszym a drugim wystąpieniem liczby 4 znajduje się osiem innych liczb $5, 3, 6, 7, 8, 1, 9, 5$.

Napisz predykat `lista(N, X)`, który jest spełniony, jeśli dla danego `N`, lista `X`:

- ma długość `2*N`,
- każda liczba od 1 do `N` występuję na niej dokładnie dwa razy,
- między dwoma kolejnymi wystąpieniami tej samej liczby jest parzysta liczba innych liczb.

Można wykazać, że jest `N!` różnych list `X` spełniających warunek `lista(N, X)`.

Przykład

```
?- lista(3, X).
X = [1, 1, 2, 2, 3, 3] ;
X = [1, 1, 2, 3, 3, 2] ;
X = [1, 2, 2, 1, 3, 3] ;
X = [1, 2, 2, 3, 3, 1] ;
X = [1, 2, 3, 3, 2, 1] ;
X = [1, 2, 3, 1, 2, 3] ;
false.
```

Uwaga 0

Zwróć uwagę, że w powyższych odpowiedziach, jeśli na liście X pojawia się po raz pierwszy liczba $k \in \{1, 2, \ldots, n\}$, to na wcześniejszych pozycjach listy X pojawiły się już wszystkie liczby $1, 2, \ldots, k-1$ (jeden lub dwa razy).

Odpowiada to numerowaniu kolejnych nieodwiedzonych wcześniej przecięć kolejnymi numerami $1, 2, \ldots, n$ (pierwsze pojawienia się liczb tworzą ciąg kolejnych liczb $1, 2, ..., n$).

Jeśli odpowiedzi z Twojego predykatu nie spełniają tego warunku, to zadanie może zostać uznane za rozwiązane ale otrzymasz dużo więcej rozwiązań niż $N!$ i być może nie doczekasz się na przejrzenie wszystkich odpowiedzi, już dla stosunkowo małych wartości N.

Uwaga 1

Nie każdej liście `X` spełniającej warunek `lista(N, X)`, odpowiada jakaś krzywa zamknięta o takich przecięciach.

Dla przykładu, liście `[1, 2, 3, 4, 5, 1, 4, 5, 2, 3]` nie odpowiada żadna krzywa zamknięta o pięciu przecięciach.

Uwaga 2

Pomyśl o jak najefektywniejszym znajdowaniu list spełniających warunek `lista/2`.

Tabela 1: Średnia liczba kroków wnioskowania na znalezienie kolejnej listy spełniającej warunek `lista/2`

N	N!	inf	avg
1	1	14	14.00
2	2	29	14.50
3	6	77	12.83
4	24	281	11.71
5	120	1356	11.30
6	720	8071	11.21
7	5040	56519	11.21
8	40320	453167	11.24
9	362880	4088366	11.27
10	3628800	40974845	11.29
11	39916800	451618977	11.31
12	479001600	5428949737	11.33

W tabeli 1 podano dla kolejnych `N = 1..12`, liczbę kroków wnioskowania, uzyskaną w odpowiedzi na cel `time((lista(N, _), fail))`, i średnią liczbę kroków na jedno z `N!` rozwiązań.

Jak widać średnia liczba kroków wnioskowania na wygenerowanie kolejnego rozwiązania (dla $N > 3$) jest poniżej 12.

Przygotuj taką tabelkę dla Twojej implementacji predykatu `lista/2`. Jaką uzyskałeś średnią liczbę kroków wnioskowania na jedno rozwiązanie?