Tölvunarfræðideild
Háskólinn í Reykjavík

Heimaverkefni 3
Vorönn 2025

Forritunarmál
Heimaverkefni 3: Prolog (50 stig/points)

Í eftirfarandi verkefnum megið þið EKKI nota innbyggð vensl nema að það sé sérsaklega tekið fram

In the following projects, you are NOT allowed to use built-in relations unless it is specifically stated.

1) **(12 points)** Í tiltekinni tölvunarfræðideild er listinn yfir kennara og námskeið þessi / *In a particular computer science department, the list of teachers and courses is as follows:*

Instructor	**Course**	**Abbreviation**
hrafn	Programming Languages	**prla**
yngvi	Compilers	**comp**
yngvi	Deep Learning	**deep**
harpa	Discrete Mathematics	**dmath**
stephan	Artificial Intelligence	**ai**
marta	Software Requirements and Design	**srd**
luca	Discrete Mathematics 2	**dmath2**
bjorn	Database Systems	**dbase**
kari	Programming	**prog**
kari	Computer Graphics	**cgra**
gylfi	Computer Organization	**corg**
arnar	Programming in C++	**cplus**

Eftirfarandi tafla sýnir alla nemendur í deildinni ásamt þeim námskeiðum sem þeir hafa tekið / *The following table lists all the students in the department along with the courses which the students have taken*:

Student	**prla**	**comp**	**deep**	**dmath**	**ai**	**srd**	**dmath2**	**dbase**	**prog**	**cgra**	**corg**	**cplus**
siggi	x	x	x	x						x		
joi			x		x		x				x	
gummi				x				x	x			x
sigga		x				x				x		
gunna	x	x	x				x				x	x
jona	x			x		x		x				x

Skrifið eftirfarandi **vensl** með því að nota upplýsingarnar að ofan (notið nöfn kennara, nöfn nemendan og skammstöfun á námskeiðum sem "atom") / *Write the following **relations** using the above information (use teacher's names, student's names and the course abbreviations as atoms):*

```
teaches_course(X,Y)      % Instructor X teaches course Y (a fact)
takes_course(X,Y)        % Student X takes course Y (a fact)
teaches_student(X,Y)     % Instructor X teaches student Y (a rule)
student(X)               % X is a student at the department  (a rule)
takes_courses(X,L)       % Student X takes all courses in list L (a rule)
student_list(L)          % L is a list of students (a rule)
```

Athugið eftirfarandi / *Note the following*:

- `student(X)` should NOT be implemented as a fact.
- `student_list(L)` should be implemented using the built-in relations `findall` (https://cs.union.edu/~striegnk/learn-prolog-now/html/node96.html) and `list_to_set` (https://www.swi-prolog.org/pldoc/man?predicate=list_to_set/2).

Dæmi um fyrirspurnir og svör */ Example queries and answers:*

```
?- teaches_course(yngvi, Y).
Y = comp ;
Y = deep.

?- takes_course(X, dmath).
X = siggi ;
X = gummi ;
X = jona.

?- teaches_student(hrafn, Y).
Y = siggi ;
Y = gunna ;
Y = jona.

?- teaches_student(X, siggi).
X = hrafn ;
X = yngvi ;
X = yngvi ;
X = harpa ;
X = kari ;
false.

?- student(gunna).
true.

?- student(bjossi).
false.
```

```
?- takes_courses(gummi, [dmath, dbase, prog]).
true .

?- takes_courses(X, [prla, comp, deep]).
X = siggi ;
X = gunna ;
false.
```

(You don't have to implement `takes_courses(X,L)` such that it is possible to answer a query like `takes_courses(gunna, L)`).

```
?- student_list(X).
X = [siggi, joi, gummi, sigga, gunna, jona].
```

2) **(24 points)** Skrifið eftirfarandi vensl / Write the following relations:

a) `myprefix(L1,L2)` % The list L1is a prefix of the list L2.

Dæmi / Examples:

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

?- myprefix([1,2],X).
X = [1, 2|_].
```

b) `num_elements(K,L)` % K is the number of elements in list L.

Dæmi / Examples:

```
?- num_elements(3,[1,2,3]).
true.

?- num_elements(X,[a,b,c,d]).
X=4.
```

c) `remove_element(X,L1,L2)`
% The list L2 is the result of removing exactly one instance of the element X from the list L1.

Dæmi / Examples:

```
?- remove_element(2,[1,2,2,3],L).
L = [1, 2, 3] ;
L = [1, 2, 3] ;
false.

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

?- remove_element(2,L,[1,2,3]).
L = [2, 1, 2, 3] ;
L = [1, 2, 2, 3] ;
L = [1, 2, 2, 3] ;
L = [1, 2, 3, 2] ;
false.

?- remove_element(X,[1,2,3],L).
X=1,  L=[2,3];
X=2,  L=[1,3];
X=3,  L=[1,2];
false.

?- remove_element(X,[a,b,c,a],L).
X = a, L = [b, c, a] ;
X = b, L = [a, c, a] ;
X = c, L = [a, b, a] ;
X = a, L = [a, b, c] ;
false.
```

d) `insert_element(Elem,L1,Pos,L2)`
% The list L2 is the result of putting element Elem into the list L1 in position Pos.

Dæmi / Examples:

```
?- insert_element(a,[b,c,d],1,[a,b,c,d]).
true.

?- insert_element(X,[1,3,4],2,[1,2,3,4]).
X = 2 .

?- insert_element(c,[a,b,d],3,L).
L = [a, b, c, d] .


?- insert_element(c,[a,b,d],X,[a,b,c,d]).
X = 3 .
```

e) `add_up_to(L1,L2)`
% L2 is a list of integers in which each element is the sum of the elements in list L2 upto the same position. Here, L1 is bound, i.e. the value of L1 is known when a query is made.

% Hint: Use the helper relation `add_up_to_helper(L1,L2,Acc)`, where Acc (accumulator) is the value that should be added to the head element of L1 to produce the element of L2 at the current position in the recursion.

Dæmi / Examples:

```
?- add_up_to([1,2,3,4],[1,3,6,10]).
true.

?- add_up_to([1,2,3,4],[1,2,6,10]).
false.

?- add_up_to([1,3,5,7],L).
L = [1,4,9,16].
```

f) `remove_duplicates(L1,L2)`
% The list L2 contains only unique elements from L1
% The last instance of repeated elements is kept in L2.
% Here you can use the built-in **member(X,L)** predicate to check if an element X is in the list L.
% L1 is bound when a query is made.

Dæmi / Example:

```
?- remove_duplicates([a, b, a, c, b, d], X).
X = [a, c, b, d] .
```

g) `split_even_odd(L, Even, Odd)`
% Even is a list containg elements at even indices in list L and Odd is a list containing elements at odd indices in list L. The first item in list L is at index 0.

Dæmi / Examples:

```
?- split_even_odd([a, b, c, d, e], Even, Odd).
Even = [a, c, e], Odd = [b, d] .

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

h) `merge_sorted(L1, L2, L3)`
% L3 is the (sorted) results of merging the two sorted lists L1 and L2.
% Both L1 and L2 are bound when a query is made

Dæmi / Example:

```
?- merge_sorted([1, 3, 5], [2, 4, 6], X).
X = [1, 2, 3, 4, 5, 6] .
```

3) **(8 points)**

a) Stafli / *Stack*

Útfærið stafla í Prolog (með því að nota lista) sem samanstendur af eftirfarandi venslum / *Implement a stack in Prolog (by using lists) which consists of the following relations:*

```
empty_stack(S)
```

% S is an empty stack

```
push(Elem, Stack, NewStack)
```

% NewStack is the result of pushing Elem onto the stack Stack.

```
pop(Elem, Stack, NewStack)
```

% NewStack is the result of popping Elem from the stack Stack.

```
top(Elem, Stack)
```

% Elem is the top element of the stack Stack.

Dæmi / Example:

```
?- empty_stack(S), push(10, S, S1), push(20, S1, S2),
pop(X, S2, S3), top(Top, S3).
S = [],
S1 = S3, S3 = [10],
S2 = [20, 10],
X = 20,
Top = 10.
```

b) Biðröð / *Queue*

Útfærið biðröð í Prolog (með því að nota lista) sem samanstendur af eftirfarandi venslum / *Implement a queue in Prolog (by using lists) which consists of the following relations:*

```
empty_queue(S)
```

% S is an empty queue

```
enqueue(Elem, Queue, NewQueue)
```

% NewQueue is the result of adding Elem at the back of queue Queue.

```
dequeue(Elem, Queue, NewQueue)
```

% NewQueue is the result of removing Elem from the front of queue Queue.

```
front(Elem, Queue)
```

% Elem is the front elment of queue Queue.

Dæmi / Example:

```
?- empty_queue(Q), enqueue(5, Q, Q1), enqueue(7, Q1,
Q2), dequeue(X, Q2, Q3), front(First, Q3).
Q = [],
Q1 = [5],
Q2 = [5, 7],
X = 5,
Q3 = [7],
First = 7.
```

4) **(6 points)** Þáttun / *Parsing*

Eftirfarandi samhengisfrjálsa mállýsing er gefin / The following CFG is given:

<expr> ::= <term> | <term> + <expr>
<term> ::= <factor> | <factor> * <term>
<factor> ::= <num> | (<expr>)

G.r.f. að strengur í þessu máli sé settur fram sem listi af tölum og táknum (e. symbols).

Dæmi: Segðin 5*(2+7) er sett fram sem listinn [5, *, '(', 2, +, 7, ')']

Skrifið venslin `expr(L)` (og önnur nauðsynleg vensl) þannig að L sé gild segð í samræmi við mállýsinguna. Hér er leyfilegt að nota **append** og **number** venslin.

Let us assume that a string in this language is represented as a list of numbers and symbols.

Example: The expression 5*(2+7) *is represented as the list* [5, *, '(', 2, +, 7, ')']

Write the relation `expr(L)` (and other necessary relations) *such that L is a valid expression according to the grammar. Here you can use the* ***append*** *and* ***number*** *relations.*

Dæmi / Examples:

```
?- expr([5]).
true.

?- expr([5,+,2]).
true.

?- expr([5,+]).
false.

?-  expr([5, *, '(', 2, +, 7, ')']).
true.
```

<u>Leiðbeiningar varðandi skil / *Instructions for submission*:</u>

Þið eigið að skila einni skrá inn í Gradescope, project3.pl sem inniheldur skilgreiningar á öllum Prolog venslunum.