Intelligente Systemen Programmeeropdracht 3: Predicaten definiëren in Prolog

Deadline: zaterdag 20 maart, 23:59 uur

(Deze versie is van 10 maart 2021, 22:49)

Je moet deze programmeeropdracht **zelf** en **alleen** maken. Je mag met anderen overleggen over de voors en tegens van verschillende oplossingsstrategieën en datastructuren, want daar leer je allebei van! Maar je mag niet je eigen code laten zien of met anderen delen, en ook geen code van anderen overnemen. Je mag ook geen code van internet overnemen. De uiterst simpele algemene regel is:

> *Elke letter code die je submit moet door jou zelf bedacht en geschreven zijn.*

Het is (verstandiger maar vooral) leuker `<preek>`en leerzamer`</preek>` om je eigen code te maken en je eigen "CORRECT" te verdienen op DOMjudge. Je moet je programma schrijven in Prolog, en inleveren via DOMjudge.

Inhoudsopgave

1 Opdrachtbeschrijving

Lees allereerst de handleiding voor het submitten van Prolog code op DOMjudge, die je kunt vinden op Blackboard. Er staat dingen in die je moet doen en dingen die je moet laten en als je je daaraan houdt, bespaar je ons allemaal tijd en jezelf stress.

Voor deze opdracht moet je een aantal predicaten definiëren door Prolog facts en rules te schrijven. Omdat de *practical sessions* volgens de auteurs van LPN (en mij) de belangrijkste

onderdelen van het hele boek zijn, wil ik je aansporen hiermee te oefenen door je voor deze practicumopdracht een aantal van deze opdrachten en LPN-exercises te laten doen, naast een paar andere predicaten.

Als je een predicaat moet schrijven dat ook in LPN wordt genoemd, en daar wordt geïllustreerd aan de hand van voorbeelden, moet je zulke **voorbeelden niet in je bestand opnemen**. Je moet **uitsluitend het gevraagde predicaat** in je bestand beschrijven. Bij de opdracht 'crossword' bijvoorbeeld, moet je niet facts als `word(astante, a,s,t,a,n,t,e).` in je bestand opnemen, maar alleen een aantal facts en/of rules beschrijven die het predicaat op de gevraagde manier definiëren. Ik beschrijf dat soort inputs namelijk in de testcases, en ik wil daarbij niet beperkt worden doordat ik conflicten moet vermijden met dergelijke inputs. Verder moet je alle **predicaten precies zo noemen als in deze opdracht**, want anders kan ik ze in mijn testcases niet aanroepen.

Ik heb deze tweede opdracht verdeeld in een aantal deelopdrachten. Voor elke deelopdracht schrijf je de definitie van een of meerdere predicaten door facts en rules te specificeren. Het bestand waarin je dat doet submit je bij het juiste 'problem' van DOMjudge, zodat dat predicaat los van andere predicaten wordt getest op een aantal testcases. Hierdoor krijg je gerichtere feedback van DOMjudge, dan wanneer je alle predicaten in één bestand samenvoegt en die in DOMjudge allemaal tegelijk worden getest.

2 Instructies (beperkingen)

Je moet je 'programma' ('knowledge base') natuurlijk zelf bedenken en opschrijven. Je mag erover praten met medecursisten—we bevelen dat zelfs aan, want je kunt elkaar helpen en zo zelf ook méér leren. Maar beperk je tot 'hoog niveau', en ga er zeker niet toe over code met elkaar te delen. Degene die code beschikbaar stelt is net zo 'strafbaar' als degene die de code overneemt.

Hoewel je je eigen code moet bedenken en programmeren, kun je niet om bepaalde ingebouwde predicaten van Prolog heen. Maar je mag niet onbeperkt gebruik maken van hetgeen Prolog biedt. Wat níet mag, is "`;`" als "of" gebruiken, want daar wordt code minder goed leesbaar door. Je mag ook geen cut "`!`" gebruiken. Je mag natuurlijk wél `=/2` gebruiken, en ook `\=/2`, en in het algemeen mag je de negatie `\+/1` gebruiken. Je mag Prolog laten rekenen, en dus gebruik maken van het predicaat `is/2`, en van rekenkundige functies zoals `+/2`, `-/2`, etc., en operatoren zoals `</2`, `=</2`, `>/2`, en `>=/2`. Soms is het handig om te weten of een variabele aan een getal gebonden is, daar mag je een ingebouwd predicaat voor gebruiken. Uiteraard mag je ook lijsten `[H|T]` gebruiken. Om PR2 tot een goed einde te brengen, is dit zeker afdoende. Als je andere relaties nodig hebt (dan hieronder genoemd staan), dan moet je die zelf definiëren. Je mag dan predicaten die je bij een andere deelopdracht hebt geschreven hergebruiken.

3 Deelopdrachten

Ik weet dat je met weinig moeite online een aantal van de oplossingen kunt vinden, maar probeer het alsjeblieft zelf. Het is echt veel leuker om dit uit te vogelen dan over te tikken van een website of een medestudent. Bovendien is dat fraude. Het DOMjudge probleem waar je je code moet submitten staat in de sectietitel tussen haakjes.

3.1 Verwijder 3'en ('3.1 3weg')

Hier moet je een predicaat `verwijder3/2` definiëren, zodat `verwijder3(L1,L2)` slaagt desda `L2` dezelfde lijst is als `L1`, maar dan zonder dat het getal '3' daarin nog als element voorkomt. We krijgen bijvoorbeeld de volgende interactie.

```
?- verwijder3([3,2],[2]).
true.
?- verwijder3([1,2],X).
X = [1,2].
?- verwijder3([3,3,3],[3]).
false.
```

3.2 Verwijder lijstelementen ('3.2 weg')

Schrijf een definitie van het predicaat `verwijder/3`, zodat `verwijder(L1,L2,L3)` slaagt desda `L3` een lijst is, die gelijk is aan lijst `L2`, waarin alle alle elementen uit lijst `L1` niet voorkomen. Wat we willen is bijvoorbeeld het volgende.

```
?- verwijder([],[1,2],[1,2]).
true.
?- verwijder([0,1,3],[0,1,2,3],X).
X = [2].
?- verwijder([[1,2]],[1,[1,2]],X).
X = [1].
```

Je mag, zoals eerder aangegeven, **geen ingebouwde Prolog predicaten gebruiken** (behalve de lijst `[]` natuurlijk). Als je een predicaat nodig hebt om te bepalen of een term element is van een lijst, moet je dat zelf schrijven.

3.3 Faculteit ('3.3 fac')

Voor deze opdracht schrijf je Prolog code om de faculteitsfunctie $n!$ uit te rekenen. Deze functie is voor een natuurlijk getal $n \in \{0, 1, 2, 3, \ldots\}$ gedefinieerd als

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n.$$

(Let op dat je de juiste waarde geeft als $n = 0$, zoek zelf even op wat die waarde is.) Je moet dit op twee manieren doen, namelijk zonder en met accumulator. De naïeve versie zonder accumulator is *left recursive* (very bad!). De versie met accumulator is *tail recursive*, dus de recursieve call moet de laatste goal van de rule zijn. In het bijzonder gaat het om twee predicaten `faculteitLeft/2` en `faculteitTail/2`, zodat `faculteitLeft(N,F)` en `faculteitTail(N,F)` slagen desda `F` gelijk is aan `N`!. Er is in dit geval overigens weinig te merken aan het verschil in berekeningstijd tussen beide varianten. Dat is heel anders bij de berekening van Fibonacci getallen.

3.4 Somlijst ('3.4 som')

Het is handig om de code die je schrijft voor de problemen die in secties 3.4, 3.5 en 3.6 zijn beschreven, in één bestand te stoppen, en dat steeds te uploaden. Voor 3.6 kun je namelijk gebruik maken van de code van 3.4 en 3.5.

Schrijf in Prolog *op twee manieren*—zonder en met accumulator—een definitie van een 'somlijst' predicaat met twee argumenten, dat slaagt wanneer het eerste argument een lijst met getallen is (voor deze test mag je een ingebouwd Prolog predicaat gebruiken), en het tweede argument de som van de getallen in die lijst. Het eerste predicaat, zonder accumulator, heet `somlijstZ/2`, het tweede heet `somlijst/2`, zodat `somlijstZ(L,S)` en `somlijst(L,S)` slagen desda `L` een lijst getallen is, en `S` de som van de getallen in `L` is. We zien dus bijvoorbeeld het volgende (waar `somlijst` staat, moet je ook `somlijstZ` kunnen invullen).

```
?- somlijst([1,2,3,4,5],N).
N = 15.
?- somlijst([0,1,-2],Bladiebla).
Bladiebla = -1.
?- somlijst([0,1,-1,Pomtiedom],Pomtiedom).
false.
```

Je mag een ingebouwd Prolog predicaat gebruiken om te testen of iets een getal is, en je mag natuurlijk `is/2` gebruiken.

3.5 Sublijst ('3.5 sub')

Schrijf een Prolog programma dat het predicaat `sublijst/2` definieert, zodat `sublijst(L,X)` slaagt desda `L` en `X` lijsten zijn, en `L` een sublijst is van `X`. Het moet **niet** zoals het `sublist/2` predicaat in hoofdstuk 6 van LPN, maar **wel** zodanig dat alle *elementen* uit de eerste lijst in de gegeven volgorde, maar *niet noodzakelijk aaneengesloten*, voorkomen in de tweede lijst. Het predikaat kan bijvoorbeeld gebruikt worden om alle mogelijke sublists van een lijst te geven:

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

3.6 Een sublijst met een bepaalde som ('3.6 subsom')

Schrijf een Prolog programma dat het predikaat `subsom/3` definieert, zodat `subsom(L,N,X)` slaagt desda `L` en `X` lijsten zijn, en `X` een sublijst van `L` is met als som van de getallen `N`. Dus:

```
?- subsom([1,2,3,4],5,[1,4]).
true.
?- subsom([1,2,3,4],5,[2,4]).
false.
```

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

Je mag je eigen predicaten `somlijst/2` (sectie 3.4) en `sublijst/2` (sectie 3.5) gebruiken. Je moet de code van de verschillende predicaten dan in één bestand kopiëren, want ik heb nog niet uitgevonden hoe je ook in het geval van Prolog meerdere bestanden kunt submitten.

3.7 Permutaties ('3.7 perm')

Schijf een Prolog programma waarin je **op twee manieren** een 'permutatie' predicaat definieert, `permutatieI/2` en `permutatieV/2`, zodat `permutatieI(P,L)` en `permutatieV(P,L)` slagen desda `P` en `L` lijsten zijn, en `P` een permutatie is van de elementen van `L`. Voor beide willen we de volgende interacties waarnemen.

```
?- permutatie([1,3,2],[1,2,3]).
true.
?- permutatie([1,2,2],[1,2,3]).
false.
?- permutatie(P,[a,b,c]).
P = [a, b, c] ;
P = [b, a, c] ;
P = [b, c, a] ;
P = [a, c, b] ;
P = [c, a, b] ;
P = [c, b, a] ;
false.
?- permutatie([1,2,3],L).
L = [1, 2, 3] ;
L = [2, 1, 3] ;
L = [3, 1, 2] ;
L = [1, 3, 2] ;
L = [2, 3, 1] ;
L = [3, 2, 1] ;
<...hangt...>
Action (h for help) ? abort
% Execution Aborted
```

Het resultaat van die laatste query geeft aan dat je niet hoeft te kunnen bepalen van welke lijsten `L` een gegeven lijst de permutatie is. Na de zes genoemde lijsten zijn de mogelijkheden op, maar dat weerhoudt Prolog er niet van om dóór te zoeken. Andersom moet je wel kunnen bepalen wat van een gegeven lijst alle permutaties zijn, zoals geïllustreerd door de voorlaatste query

```
?- permutatie(P,[a,b,c]).
```

Het verschil tussen de predicaten `permutatieI/2` en `permutatieV/2` is dat de eerste gebruik moet maken van een predicaat `invoeging/3` en het tweede van een predicaat `verwijdering/3`.

Deze beide predicaten moet je ook zelf schrijven en precies deze naam geven, want ze worden afzonderlijk getest. Er moet gelden dat `invoeging(X,L1,L2)` slaagt desda `L1` en `L2` lijsten zijn, en `L2` gelijk is aan `L1`, met `X` daaraan toegevoegd. Voor `verwijdering/3` geldt, *mutatis mutandis*, hetzelfde, maar dan is `X` verwijderd.

3.8 Een lijst van ... tot ... ('3.8 vanTot')

Schrijf een Prolog programma dat het predicaat `vanTot/3` definieert, zodat `vanTot(L,H,P)` slaagt desda `L` en `H` gehele getallen zijn (voor deze test mag je een ingebouwd Prolog predicaat gebruiken), en `P` een lijst is met als elementen alle gehele getallen tussen `L` en `H` (inclusief).

```
?- vanTot(1,8,CormenNorvig).
CormenNorvig = [1, 2, 3, 4, 5, 6, 7, 8]
?- vanTot(8,1,Obamovidansiwarkimeridistanstiva).
Obamovidansiwarkimeridistanstiva = [8, 7, 6, 5, 4, 3, 2, 1]
?- vanTot(1,3.5,NormenCorvig).
false.                  % want 3.5 is geen geheel getal
?- vanTot(1,X,[1,2,3]).
false.                  % want X is geen geheel getal
?- vanTot(3,3,L).
L = [3] ;
false.
```

3.9 Een lijst met een bepaalde som ('3.9 lijst')

Schrijf in Prolog een definitie van het predicaat `lijstsom/2` zodat `lijstsom(L,S)` slaagt desda `L` een lijst opeenvolgende gehele getallen is, beginnend met 1, met als som `S`. Je kunt hier het predicaat `vanTot/3` van de vorige deelvraag gebruiken, schrijf de code voor deze opdracht dan in dat bestand, bij de code die je voor dat probleem al hebt gesubmit. Je mag verder het predicaat `is/2` gebruiken, en ook rekenkundige operatoren zoals `sqrt/1`, `floor/1`, en `*/2` (of de bijbehorende predicaten).

```
?- lijstsom(L,6).
L = [1, 2, 3] ;
false.
?- lijstsom(L,45).
L = [1, 2, 3, 4, 5, 6, 7, 8, 9] ;
false.
?- lijstsom(L,46).
false.
```