

Instructies:

Dit tentamen bestaat uit 12 opgaven. Er zijn in totaal 100 punten te verdienen. De maximaal toegestane tijd voor dit tentamen is 120 minuten. Geef korte duidelijke antwoorden in het Nederlands of Engels en schrijf leesbaar. Gebruik hiervoor alleen het uitgereikte examenpapier en schrijf op elk blad je naam en studentnummer. Code hoeft niet becommentarieerd te worden, maar moet wel vrij zijn van syntax errors en singleton warnings. Ook ongewenste resultaten bij backtracken moeten voorkomen worden. Built-in predicaten mogen alleen gebruikt worden indien ze genoemd worden bij de opgave. Standaard operatoren als `=`, `>`, `is` en `!` mogen altijd gebruikt worden, evenals de predicaten `call/0`, `true/0` en `fail/0`.

Opgave 1: (10 punten)

Definieer een predicaat `diagonal/1` dat een diagonale lijn van sterretjes print. (De richting van de diagonaal is niet van belang.) Maak geen gebruik van andere built-ins dan het predicaat `write/1`, VB:

```
?- diagonal(4).
*
 *
  *
   *
true.
```

Uitwerking:

```
diagonal(N):-
    diagonal(N, 0).           % write from 0 to N-1 so we can use
                              % M for the whitespace.

diagonal(N, N):-!.          % stop looping if backtracking

diagonal(N, M):-
    white(M),                 % tab(M) would normally do this
    write('*'),
    nl,
    NewM is M + 1,
    diagonal(N, NewM).

white(0):-!.                 % stop looping if backtracking

white(N):-
    write(' '),
    NN is N - 1,
    white(NN).
```

Opgave 2: (10 punten)

Slagen de volgende queries? Geef in het geval van slagen het eerste antwoord. Geef in de andere gevallen kort aan waar de query precies over struikelt.

- a) `?- [A|A] = [A].`
- b) `?- 'Hi'(bye) =.. B.`
- c) `?- [a, b] =.. C.`
- d) `?- [1, 2, 3, 4] = [X|Y], D = [Y|[X]].`
- e) `?- 5*2 = X, Y = X*3, Z = 5*2*3, Y is Z.`

Uitwerking:

- a) `?- [A|A] = [A].`
`A = [].`
- b) `?- 'Hi'(bye) =.. B.`
`B = ['Hi', bye].`
- c) `?- [a, b] =.. C.`
`C = ['.', a, [c]].`
- d) `?- [1, 2, 3, 4] = [X|Y], D = [Y|[X]].`
`X = 1,`
`Y = [2, 3, 4],`
`D = [[2, 3, 4], 1].`
- e) `?- 5*2 = X, Y = X*3, Z = 5*2*3, Y is Z.`
`fail. >>> Y en Z zijn allebei 5*2*3, maar 'is'`
`rekent de rechterkant uit en de linker niet, dus`
`zijn ze niet gelijk.`

Opgave 3: (10 punten)

Definieer je eigen negatie predicaat genaamd *niet* die zich gedraagt als de built-in operatoren `\+` en *not*. Je mag hier uiteraard geen gebruik van maken. Zorg vervolgens dat je het predicaat zonder haakjes als prefix operator kan gebruiken en leg kort uit hoe je predicaat werkt.

```
?- niet true.  
fail.  
  
?- niet niet true.  
true.
```

Uitwerking:

```
:-
    op(200, fy, niet).

niet(X):-
    call(X),
    !,
    fail.

niet(_).

% uitleg: zie Bratko & Endriss.
```

Opgave 4: (8 punten)

Gegeven dat de volgende definitie van het predicaat *bs/2* geladen is:

```
bs(N, small):-
    N =< 555,
    !.

bs(_, big).
```

Hoe zal prolog reageren op de volgende queries? Geef het eerste antwoord.
Leg kort uit bij niet slagen.

- a) ?- bs(-1000, X), bs(1000, Y).
- b) ?- member(X, [74, 930]), bs(X, Y), Y = big.
- c) ?- bs(X, big), member(X, [-123, 836]).
- d) ?- bs(X, small), member(X, [-123, 836]).

Uitwerking:

```
?- bs(-1000, X), bs(1000, Y).
X = small,
Y = big.

?- member(X, [74, 930]), bs(X, Y), Y = big.
X = 930,
Y = big.

?- bs(X, big), member(X, [-123, 836]).
X = -123.

?- bs(X, small), member(X, [-123, 836]).
ERROR: =</2: Arguments are not sufficiently instantiated
```

Opgave 5: (10 punten)

Definieer een predicaat *squares/2* dat voor een gegeven lijst getallen de lijst van kwadraten teruggeeft, VB:

```
?- squares([3, 1, 5, 4], S).  
S = [9, 1, 25, 16].
```

Uitwerking:

```
squares([], []).  
  
squares([H|T], [SH|ST]):-  
    SH is H * H,  
    squares(T, ST).
```

Opgave 6: (10 punten)

Gegeven de volgende operator definities:

```
:-      op(400, xfx, alpha),  
        op(300, yfx, beta),  
        op(500, fy, gamma),  
        op(200, xf, delta).
```

Hoe zal prolog reageren op de volgende queries? Wees zeer precies met de eventuele haakjes in het antwoord.

- a) `?- A = (p alpha (q beta r)) beta s.`
- b) `?- P beta R = 1 delta beta 2 beta 3.`
- c) `?- X alpha Y = 1 alpha 2 alpha 3.`

Vragen:

- d) Wat is de principal operator van de term:

```
gamma (1 alpha 2) alpha 3 delta
```

- e) Wat is de langste zin die te maken is met de operatoren alpha en delta en een vrij te kiezen aantal getallen?

Uitwerking:

```
?- A = (p alpha (q beta r)) beta s.  
A = (p alpha q beta r)beta s.
```

```
?- P beta R = 1 delta beta 2 beta 3.  
P = 1 delta beta 2,  
R = 3.
```

```
?- X alpha Y = 1 alpha 2 alpha 3.  
ERROR: Syntax error: Operator priority clash  
ERROR: X alpha Y = 1 alpha  
ERROR: ** here **  
ERROR: 2 alpha 3 .
```

```
gamma (1 alpha 2) alpha 3.  
Principal operator is gamma want deze heeft de hoogst precedence.
```

De langste zin is: 1 delta alpha 2 delta

Want alpha is heeft de hoogste precedence en wordt dus principal operator. Ook is deze niet associatief dus kan alpha maar 1 keer voorkomen. De linker en rechter term kunnen dus alleen nog van delta gebruik maken. Dit kan maar 1 keer per term omdat deze operator niet associatief is.

Opgave 7: (6 punten)

Definieer zonder recursie te gebruiken het predicaat *naast/3* dat slaagt als twee elementen X en Y naast elkaar in een lijst L voorkomen. Je mag voor deze opgave indien nodig gebruik maken van de built-ins: *member/2*, *append/3*, *select/3* en *length/2*, VB:

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

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

```
?- naast(a, c, [a, b, c, d]).  
fail.
```

Uitwerking:

```
naast(X, Y, List):-  
    append(_, [X, Y|_], List).
```

```
naast(X, Y, List):-  
    append(_, [Y, X|_], List).
```

Opgave 8: (6 punten)

Gegeven dat de volgende definitie van het predicaat *what/2* geladen is:

```
what(A, [A|B]):-  
    C is A - 1,  
    what(C, B).
```

```
what(0, []).
```

Hoe zal prolog reageren op de volgende query?

a) `?- what(5, X).`

Vragen:

b) Geef 2 mogelijke verbeteringen van dit programma aan als het gebruikt moet worden met een getal als 1^e argument en een variabele als 2^e argument.

c) Wat is nu de uitkomst van dezelfde query?

Uitwerking:

Deze query komt in een eindeloze loop terecht tot hij uit zijn geheugen loopt met een melding als: *out of local stack*.

Het programma is te repareren door de stopclause bovenaan te zetten en een cut of check toe te voegen zodat backtracken ook geen loop meer oplevert.

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

Opgave 9: (5 punten)

Leg kort uit wat de 'Closed World Assumption' inhoudt. Geef aan wanneer dit principe problemen op kan leveren.

Uitwerking:

Zie Bratko en Endriss. Een mogelijk antwoord kan zijn: De Closed World Assumption houdt in *dat aangenomen wordt dat alle relevante feiten in de prolog database staan*. Dus als iets waar is dan staat het in de database. Dit kan bv. problemen opleveren in combinatie met het gebruik van de negatie operator. Als een feit niet bewezen kan worden is de negatie ervan waar. Maar *kan het feit niet bewezen worden omdat het niet waar is of omdat de database incompleet is?* Het eenduidige antwoord van prolog zou dus in sommige gevallen beter 'onbeslist' kunnen zijn. Hier moet rekening mee gehouden worden bij het programmeren.

Opgave 10: (10 punten)

Definieer een predicaat *factorial/2* dat voor een gegeven positieve integer N de faculteit: N! teruggeeft. NB: 4! = 1 x 2 x 3 x 4. en 0! = 1 (per definitie). Zorg voor een efficiënte implementatie dus maak gebruik van tailrecursie indien mogelijk.

```
?- factorial(5, F).  
F = 120.
```

Uitwerking:

```
% deze clause zorgt ervoor dat we ons predicaat als functie  
% kunnen gebruiken (ter educatie: niet vereist in antwoord)  
:- arithmetic_function(factorial/1).
```

```
factorial(N, F):-  
    N >= 0,  
    integer(N),  
    fac(N, 1, F).
```

```
fac(0, F, F):-  
    !.
```

```
fac(N, X, F):-  
    X1 is N * X,  
    N1 is N - 1,  
    fac(N1, X1, F).
```

Opgave 11: (5 punten)

Leg het verschil uit tussen de operatoren '=' , 'is' en ':=='.

Uitwerking:

Zie Bratko en Endriss. Een mogelijk antwoord kan zijn:

De operator '=' voert matching uit. Er wordt gekeken of twee termen gelijk aan elkaar zijn of gemaakt kunnen worden doormiddel van substitutie.

De operator 'is' rekt de term aan zijn rechterkant rekenkundig uit en voert daarna matching uit tussen links en rechts.

De operator ':==' rekt de termen aan beide kanten rekenkundig uit en kijkt dan of ze gelijkwaardig zijn.

Opgave 12: (10 punten)

Gegeven dat de volgende definitie van het predicaat *effect/3* geladen is (inclusief hulppredicaten):

```
effect(A, B, _C):-
    call(B),
    asserta(this_is(A)),
    fail.

effect(_A, _B, C):-
    assertz(this_is(stop)),
    what_is_this(C).

what_is_this(A):-
    retract(this_is(B)),!,
    wit(A, B).

wit([], stop):-
    !.

wit([A|B], A):-
    what_is_this(B).
```

Hoe zal prolog reageren op de volgende query?

a) `?- effect(X, member(X, [a, b, c]), Y).`

Vragen:

- b) Welk built-in predicaat lijkt qua gedrag behoorlijk op *effect/3*?
- c) Hoe kan de code aangepast worden, zodat het gedrag gelijk wordt?
- d) Leg kort (conceptueel) uit wat *effect/3* doet en hoe het predicaat dit gedaan krijgt.

Uitwerking:

a) `Y = [c, b, a].`

b) Effect werkt bijna als de built-in *findall/3*.

c) Om identiek gedrag te krijgen moet de statement `asserta` omgezet `assertz`.

d) Effect zoekt alle antwoorden op een goal (in dit geval `member`). Deze slaat hij telkens op met `assert` voordat hij faalt en het volgende antwoordt zoekt via backtracken. Als alle antwoorden opgeslagen zijn haalt hij deze op met een loopje dat de antwoorden meteen ook weer uit de database verwijdert (`retract`).