Prolog tentamen: 31 Mei 2012

De eindbeoordeling is "het aantal punten + 10" gedeeld door 10.

Vraag:	1	2a	2b	3a	3b	4a	4b	Total
punten:	15	10	15	15	10	15	10	90

Veel succes!

Vraag 1: Unificatie (15 punten)

Geef voor elk van de volgende goals aan of ze slagen of falen. Geef bij slagende goals aan wat de resulterende variabele-binding(en) is, geef bij falende goals een korte uitleg.

```
1. ?- [a,b|C]=[A,B,c].
2. ?- [a,A]=[b,a].
3. ?- [a|[b,c]]=[a,b,c].
4. ?- [a,[b,c]]=[a,b,c].
5. ?- [a,X]=[X,b].
6. ?- [a|[]]=[X].
7. ?- [a,Z,X,c]=[A,B,Y].
8. ?- [H|T]=[[a,b],[c,d]].
9. ?- [[X],Y]=[a,b].
10. ?- [H|T]=[a].
11. ?- X =.. [b,Y,[1,2]].
12. ?- X =.. [X1,X2].
13. ?- X is 1, Y is X-1, Y < 3.
14. ?- Y < 3, X is 1, Y is X-1.
15. ?- X=3, Y=X-1, Z is Y.
```

Vraag 2: Lijsten en programeren (25 punten)

Gegeven de volgende representaties:

1. voor een verbinding van A naar B met afstand X gebruiken we een term van de vorm "verbinding(VanA,NaarB,Afstand)". De verbinding van "amsterdam" naar "utrecht" met afstand 45 wordt weergegeven als "verbinding(amsterdam, utrecht, 45)".
2. voor een route gebruiken we een lijst. De elementen van de lijst zijn termen verbindingen zoals hierboven onder (1) is beschreven.

Voorbeeld:

```
Route = [verbinding(a,b,10), verbinding(b,c,20)].
```

(2a) Programma (10 punten)

Schrijf een predicaat "valide-route(Lijst)" dat slaagt indien de Lijst een correcte route weergeeft en anders faalt. Correcte route wil zeggen dat de verbindingen aansluitend zijn.

Voorbeelden:

```
% route a-b-c
?- valide-route([verbinding(a,b,10), verbinding(b,c,20)]).
Yes

% de verbindingen in de route sluiten niet aan, immers eerste
% verbinding is naar b, en het tweede verbinding is vanuit c.
?- valide-route([verbinding(a,b,10), verbinding(c,b,20)]).
No
```

(2b) Programma (15 punten)

Schrijf een predicaat "reisafstand(Route, [Vertrek, Aankomst, Afstand])" dat gegeven een route (Route) het beginpunt van de route (Vertrek), eindpunt van de route (Aankomst) en de afstand (Afstand) van beginpunt naar eindpunt via de gegeven route bepaalt.

Voorbeelden:

```
?- reisafstand([[a,b,10], [b,c,20], [c,d,30]],
                              [Vertrek, Aankomst, Afstand]).
Vertrek = a
Aankomst = d
Afstand = 60

?- reisafstand([[a,b,10]], [Vertrek, Aankomst, Afstand]).
Vertrek = a
Aankomst = b
Afstand = 10

?- reisafstand([[a,b,10], [c,d,30]], [Vertrek, Aankomst, Afstand]).
No.
```

Vraag 3: Meta-Interpreters (25 punten)

Gegeven is de onderstaande Prolog code:

```
answer(Conclusion) :- fact(Conclusion).
answer(Conclusion) :- rule(Condition,Conclusion),
                      answer(Condition).

rule(roadworks(Someday), trafficjam(Someday)).
rule(snow(Someday), trafficjam(Someday)).
rule(rain(Someday), delays(Someday)).

fact(roadworks(today)).
fact(snow(tomorrow)).
```

(3a) Antwoord van het programma (15 punten)

Wat zal Prolog als antwoorden geven op de vraag:

```
?- answer(trafficjam(When)).
```

Leg uit hoe je aan dit antwoord komt.

(3b) Cut (10 punten)

We veranderen nu de tweede clause voor het answer-predicaat als volgt (de eerste clause blijft onveranderd):

```
answer(Conclusion) :- rule(Condition,Conclusion),
                      answer(Condition), !.
```

Wat zal Prolog nu als antwoorden geven op de vraag:

```
?- answer(trafficjam(When)).
```

Leg uit hoe je aan dit antwoord komt.

Vraag 4: Grammar (25 punten)

Gegeven zijn de volgende DCG:

```
expr --> var,[=],expr.
expr --> basicexpr, addterm.
expr --> basicexpr, factorterm.
expr --> basicexpr.
basicexpr --> [1].
basicexpr --> [2].
basicexpr --> [3].
varx --> [x].
vary --> [y].
addterm --> [plus], vary.
addterm --> [plus], basicexpr.
factorterm --> [maal], vary.
factorterm --> [maal], basicexpr.
```

Deze DCG kan gebruikt worden voor het genereren en testen van bepaalde expressies, namelijk een vergelijking bestaande uit de elementen 1, 2, 3, maal, plus, en twee variabelen 'x' en 'y'. De plus en de maal zijn hierbij infix operatoren. Een valide vergelijking is bijv. x = 1 plus 2, een niet valide expressie is y = 1 1 plus.

Enkele voorbeelden van het gedrag:

```
?- expr([x, =, 1, plus, 2],[]).
Yes.
?- expr([x, =, 3, maal, y],[]).
Yes.
?- expr([x, =, 1],[]).
Yes.
% alleen de getallen 1,2 en 3 zijn valide getallen.
?- expr([x, =, 4],[]).
No.
?- expr([x, =, 1, 1],[]).
No.
?- expr([x, =, plus, 1, maal, y],[]).
No.
```

(4a) DCG (15 punten)

Breid bovenstaande DCG regels uit zodat er een parse tree opgebouwd wordt. Hiervoor kan je een nieuw argement toevoegen aan een DCG regel. Gebruik in de parse tree de structuren is(..,..), maal(..,..), en plus(..,..), en verder de getallen 1,2, en 3. Je kan ervan uitgaan dat plus en maal gelijke precedenties hebben, en dat de input simpel van links naar rechts geevalueerd wordt. x = 2 plus 3 heeft de bijbehorende parse tree is(x, plus(2,3)). Voorbeeld:

```
?- expr(Tree,[x,=, 2, plus, 3],[]).
Tree = =is(x, plus(2, 3));
Yes.
```

Vraag 4b (10 punten)

Ga er van uit dat de parse tree bestaat uit de volgende structuren en terminals: is(..,..), maal(..,..), plus(..,..), 1, 2, 3, x, y. (zoals bij vraag 4a). Schrijf een programma *meaning* dat de waarde van de expressie uitrekent, gegeven de parse tree. (meaning(+ParseTree,-Meaning)). Voorbeeld:

```
?- expr(Tree,[x, is, 2, maal, 3],[]), meaning(Tree,Value).
Tree = is(x,maal(2, 3)),
Value = 6;
No.
```

Als in de vergelijking de variable 'y' voorkomt, vraag de gebruiker om een input van de waarde voor 'y', en geef de waarde van 'x' afhankelijk van 'y' terug.

Einde tentamen