Tentamen Intelligente Systemen (B3IS)

- Het gebruik van boeken, aantekeningen, rekenmachines of andere bronnen is **niet toegestaan**.
- Je kunt 45 punten verdienen. Je cijfer is $1 + \frac{\text{aantal punten}}{5}$.
- **Antwoord bondig!** Ik trek punten af voor overbodige uitweidingen.

1. **Logica**

(a) i. (3 punten) Maak een waarheidstabel om te onderzoeken of de volgende uitspraak waar is.

$$((p \wedge q) \rightarrow \neg r), (p \vee \neg q), (\neg q \rightarrow p) \models \neg r.$$

Antwoord: Moet geen probleem zijn, 8 rijen.

ii. (1 punt) Is de uitspraak waar?

Antwoord: Nee.

iii. (1 punt) Leg uit hoe je antwoord bij ii volgt uit je waarheidstabel van vraag i. Nummer, indien gewenst, de kolommen en rijen in de waarheidstabel en verwijs naar deze nummers in je uitleg.

Antwoord: Hier moet worden gewezen op tenminste één rij in de waarheidstafel waarin de premissen waar zijn, en de conclusie onwaar.

(b) (4 punten) Gegeven is de volgende verzameling Γ met daarin zes disjuncties, die het resultaat zijn van herschrijven naar CNV, Skolemiseren, en verwijderen van universele kwantoren.

$$\Gamma = \{C(a), \neg C(w) \vee D(w), \neg L(v,y) \vee \neg D(z) \vee \neg K(y,z),$$
$$L(g(b),b) \vee \neg D(f(b)), L(g(x),x) \vee D(f(x)), K(b,a) \vee K(c,a)\}$$

Voor het gemak zijn de variabelen in verschillende disjuncties al 'standardized apart', dus elke variabele komt maar in één disjunctie voor: $v, \ldots, z$ zijn (universeel gekwantificeerde) variabelen, a, b en c zijn constanten, en f en g zijn functiesymbolen. Geef een resolutie-afleiding die bewijst dat $\Gamma \models K(c,a)$. Tip: Zet alle benodigde formules genummerd *onder elkaar* (dus *niet naast elkaar*), en geef dan steeds een volgende (genummerde) formule die je met de resolutie-regel afleidt uit twee voorgaande formules. Geef aan uit *welke* twee formules je een formule afleidt, en welke substitutie je hebt gebruikt om literalen te laten matchen. Als je factoring doet, geef dat dan ook aan.

Antwoord:

1	$C(a)$	element van Γ
2	$\neg C(w) \vee D(w)$	element van Γ
3	$\neg L(v,y) \vee \neg D(z) \vee \neg K(y,z)$	element van Γ
4	$L(g(x),x) \vee D(f(x))$	element van Γ
5	$L(g(b),b) \vee \neg D(f(b))$	element van Γ
6	$K(b,a) \vee K(c,a)$	element van Γ
7	$\neg K(c,a)$	negatie van de conclusie
8	$K(b,a)$	resolvent van 6 en 7, geen substitutie
9	$D(a)$	resolvent van 1 en 2, substitutie $\{w/a\}$
10	$\neg L(v,y) \vee \neg K(y,a)$	resolvent van 3 en 9, $\{z/a\}$
11	$\neg L(v,b)$	resolvent van 8 en 10, $\{y/b\}$
12	$L(g(b),b) \vee L(g(b),b)$	resolvent 4 en 5, $\{x/b\}$
13	$L(g(b),b)$	factoriseren 12
14	$\square$	resolvent 11 en 13, $\{v/g(b)\}$

2. **Classical Planning**

(a) Als we PDDL gebruiken om een planningsprobleem te specificeren, resulteert dit in een bepaalde gerichte graaf waarin state-space search planningsalgoritmen (forward of backward) een plan zoeken.

i. (1 punt) Wat is een knoop in zo'n graaf?

> **Antwoord:** Een knoop is een 'state' of toestand, officieel "a conjunction of fluents that are ground, functionless atoms" (AIMA, p. 367).

ii. (1 punt) Wat representeert een kant (s, t) van knoop s naar knoop t in zo'n graaf?

> **Antwoord:** Een (gerichte) kant (s, t) is een actie die wordt uitgevoerd in de toestand s die de precondities van de actie vervult, naar toestand t waar de postcondities (eventueel negatief) waar worden.

iii. (2 punten) Als een state-space search algoritme een plan vindt, hoe ziet dat er dan uit?

> **Antwoord:** Het is een totale ordening van activiteiten, die je van de initiële naar de doeltoestand brengen.

(b) Een partial-order plan (POP) is een partiële ordening van activiteiten, die het resultaat is van een zoekproces in een *andere* gerichte graaf dan die van state-space search.

i. (1 punt) Wat is het voordeel van een POP boven een plan dat state-space search vindt?

> **Antwoord:** Het legt niet meer vast dan nodig, 'least commitment'.

ii. (2 punten) Beschrijf nauwkeurig hoe de eerste knoop in deze graaf, waar het zoekproces mee begint, eruit ziet.

> **Antwoord:** Het is een leeg plan, met alleen de initiële en de doeltoestand, en niets er tussenin.
> Deze vraag is door iedereen (op énén na) fout geantwoord. Men heeft 'deze graaf' geïnterpreteerd als het POP van deelvraag (b).i, en niet—zoals bedoeld—als de "*andere* gerichte graaf dan die van state-space search" zoals in de vraagstelling bij (b) genoemd.

iii. (2 punten) Als het PO-planningsalgoritme een bepaalde knoop s in deze graaf evalueert, wat kenmerkt dan de knopen waar naartoe vanaf s kanten lopen?

> **Antwoord:** Hier heeft niemand een punt gekregen, als gevolg van de verkeerde interpretatie die ik bij het antwoord op vraag (b).ii beschreef. Ik heb deze beide deelvragen niet meegerekend, dus het puntentotaal verminderd met 4 punten (2 voor ii. en 2 voor iii.).

3. **Plannen met resources**

(a) (1 punt) Beschouw een partial-order plan voor het uitvoeren van activiteiten die een zekere hoeveelheid tijd kosten. Als je zo'n POP als graaf weergeeft, hoe vind je daarin dan een kritiek pad?

> **Antwoord:** Tel voor elk pad van de start naar het eind van het plan de lengtes op van de duur van de activiteiten op het pad. Een pad dat de maximale lengte geeft (hiervan kunnen er meerdere zijn), is een kritiek pad.

(b) (1 punt) Wat is er 'kritiek' aan een kritiek pad?

> **Antwoord:** Als een activiteit op het kritieke pad langer (of korter) duurt dan verwacht, duurt het hele project langer (of korter) dan verwacht. Voor alle activiteiten die niet op een kritiek pad liggen is dit niet het geval.

(c) (3 punten) (Kijk vast naar vraag d!) Geef een concrete instantie van zo'n (uitsluitend temporeel) roosteringsprobleem, bepaal het kritieke pad, en leidt de 'earliest start' en 'latest start' schedules af.

Antwoord: Je moet een instantie maken waarvoor geldt dat er in het earliest start schedule twee activiteiten parallel plaats vinden, zodat we bij deelvraag (d) resources kunnen toevoegen die niet voldoende zijn om die twee activiteiten tegelijkertijd te laten plaats vinden. Neem bijvoorbeeld een startactiviteit a_1, twee activiteiten a_2 en a_3 die pas kunnen starten als a_1 is afgelopen, en een eindactiviteit a_4 die pas kan 'starten' als a_2 en a_3 klaar zijn. Elke activiteit kan bijvoorbeeld 10 tijdseenheden duren. Beide paden $a_1 - a_2 - a_4$ en $a_1 - a_3 - a_4$ zijn dan kritiek. Je kunt ook a_2 en a_3 verschillende lengte geven, dan is één van beide paden niet meer kritiek, gewoon voor de lol. Laten we dat doen: a_3 kost niet 10 maar 15 tijdseenheden. Nu is alleen het pad $a_1 - a_3 - a_4$ kritiek, met een lengte van 35. Activiteit a_2 heeft nu een slack van 5. Het ES schedule s_e is nu: $s_e(a_1) = 0$, $s_e(a_2) = 10$, $s_e(a_3) = 10$, $s_e(a_4) = 25$, en het LS schedule s_ℓ is hetzelfde voor alle activiteiten op het kritieke pad, alleen $s_\ell(a_2) = 15$.

(d) (2 punten) Breid je probleeminstantie van vraag c uit met één of meerdere resources die door activiteiten gebruikt worden. Zorg ervoor dat deze instantie illustreert dat, wanneer er resources worden toegevoegd, het earliest start schedule van de kritieke pad methode ongeldig kan worden.

Antwoord: Zoals gezegd, we voegen een resource r toe, en specificeren dat a_2 en a_3 allebei 1 unit van resource r gebruiken. De capaciteit van resource r is ... drumroll ... 1! Nu is—zoals gevraagd—het ES schedule van deelvraag (c) niet meer geldig, want daarin zijn a_2 en a_3 tegelijkertijd geroosterd, en dat kan niet, want daarvoor zijn niet genoeg units van resource r beschikbaar.

(e) (2 punten) Beschrijf een methode die je kunt gebruiken om (oplosbare) instanties van dit Resource-Constrained Project Scheduling Problem toch op te lossen—dat hoeft niet met de kortste makespan, en ook niet in een polynomiale hoeveelheid tijd. Beargumenteer wel hoe je methode een oplosbare instantie inderdaad oplost.

Antwoord: Er zijn allerlei methoden.

- Maak een topologische ordening van de partiële ordening in het rooster. Dit is een lineaire ordening van de activiteiten in een gerichte graaf, zodanig dat voor elke kant (u, v) in de graaf geldt dat u in de lineaire ordening voor v komt. Je kunt zo'n topologische ordening tekenen als een rijtje van alle activiteiten waarin de pijlen tussen de activiteiten alleen naar rechts toe lopen, en dus niet terug. Je kunt in polynomiale tijd zo'n topologische ordening vinden. Voor de activiteiten in deze volgorde uit. Omdat de instantie oplosbaar is, is er geen activiteit die in z'n eentje meer eenheden van een resource gebruikt dan er beschikbaar zijn, dus kunnen alle activiteiten achter elkaar worden uitgevoerd. Dit kost misschien wel véél meer tijd dan nodig, omdat sommige activiteiten misschien wél parallel zouden kunnen worden uitgevoerd.
- Je kunt ook iets minder intelligents doen, een soort brute-force zoeken in alle topologische ordeningen: beschouw achtereenvolgens alle $n!$ permutaties van alle activiteiten, en bepaal voor elke permutatie of die aan de precedentie-relaties in de input voldoet. De eerste die voldoet is een topologische ordening van de partiële ordening, en kun je als schedule gebruiken (volgens het resource-argument hierboven).
- Je kunt een Schedule Generation Scheme gebruiken, zoals het seriële SGS dat op het college is besproken. Dit voegt achtereenvolgens de activiteiten in (volgens een zekere prioriteitsvolgorde over de activiteiten, bijvoorbeeld geordend op latest finish time), in een partieel rooster, en wel op de eerste plek die mogelijk is, als alle activiteiten die eerder klaar moeten zijn al geroosterd zijn, en er geen resourcecapaciteiten worden overschreden.

- Je kunt Precedence Constraint Posting gebruiken, zoals op college besproken. Hierbij los je eerst alleen het temporele deel van de instantie op door het ES schedule te maken voor de activiteiten en hun precedentierelaties. Daarna bekijk je het resourcegebruik van dat ES rooster en los je achtereenvolgens de resourceconflicten op door tussen genoeg paren activiteiten die bij zo'n resourceconflict betrokken zijn precedentierelaties toe te voegen, en voor het resulterende partial order plan een nieuw ES schedule te construeren (en daarvoor weer een resourceconflict op te lossen, etc.).

4. **Kennis leren**

Bij inductief leren van kennis uitgedrukt in logica, zoeken we in een ruimte $\mathcal{H} = \{h_1, \ldots, h_n\}$ van hypothesen, naar (bijvoorbeeld) een definitie van een predicaat Goal(x). (In het voorbeeld over wel of niet op een tafeltje wachten was dit het predicaat WillWait(x): we willen uit de attributen van situatie x leren of iemand wel of niet wacht in situatie x.) In het bijzonder zoeken we een definitie die alle reeds bekende examples correct classificeert, en bovendien nieuwe examples goed voorspelt. Elke kandidaat-hypothese h_j heeft de vorm van een logische formule die $C_j(x)$ heet—dus de rij van 5 tekens $C_j(x)$ is niet zélf een formule—en we zoeken een formule zodat

$$\forall x \in \mathcal{X} \quad (\text{Goal}(x) \leftrightarrow C_j(x))$$

waar is. Hierin is $\mathcal{X}$ de verzameling examples, Goal(x) de waarde van het te leren predicaat in example x, en $C_j(x)$ een formule over de attributen van example x, bijvoorbeeld een disjunctie van conjuncties over de attributen van situatie x.

(a) (2 punten) Beschrijf in termen van de uiteenzetting hierboven nauwkeurig wat een false positive example x_p en een false negative example x_n voor een zekere hypothese genaamd h_3 zijn. Betrek in je antwoorden ook de bi-implicatie hierboven.

Antwoord: Example x_p is een false positive voor hypothese h_3, als h_3 denkt dat x_p een positive is, terwijl het dat niet is. Dat betekent dat de attributen van x_p dusdanig zijn dat $C_3(x_p)$ true is, maar Goal(x_p) false is. Dit toont aan dat één van de implicaties $(\text{Goal}(x_p) \leftrightarrow C_j(x_p))$ (wat een conjunctie van twee implicaties is) onwaar is, namelijk $(C_j(x_p) \rightarrow \text{Goal}(x_p))$ is onwaar. Example x_n is een false negative voor h_3 als x_n de andere implicatie onwaar maakt. Dan is $\neg C_3(x_n)$ waar, maar Goal(x_n) onwaar.

(b) (2 punten) Het version-space leeralgoritme houdt twee verzamelingen van hypothesen bij: De G-set van meest algemene, en S-set van meest specifieke hypothesen. Hoe initialiseert het algoritme de G-set en de S-set voordat het examples gaat evalueren? Verklaar deze initialisatie.

Antwoord: De G-set is de set van meest algemene hypothesen, dus hierin zitten hypothesen waarvan alle hypothesen die nog niet zijn verworpen specialisaties zijn. Als er nog geen enkele hypothese is verworpen (bij initialisatie van het algoritme, "voordat het examples gaat evalueren"), dan zijn alle hypothesen nog mogelijk, en ze impliceren allemaal True, dus de G-set wordt geïnitialiseert als True. Zo wordt de S-set geïnitialiseerd als False, want dan impliceert de S-set alle hypothesen.

(c) (3 punten) Wat doet het version-space leeralgoritme als het een example te verwerken krijgt dat een false negative blijkt te zijn voor hypothese S_i in de S-set? Verklaar je antwoord, en gebruik daarin ook de notie van de 'extensie' van een hypothese.

Antwoord: Dan zegt de hypothese dat het een negative is, maar eigenlijk is het een positive, dus de extensie van de definitie is niet breed genoeg. De hypothese is dus te specifiek, dus hij wordt verwijderd uit de S-set (eigenlijk naar de 'ruimte' onder de S-set verwezen), en alle directe generalisaties van S_i worden opgenomen in de S-set, als er voor elk van die generalisaties tenminste een hypothese in de G-set bestaat die er een generalisatie van is. Want anders zou zo'n generalisatie van S_i een hypothese mogelijk maken die al afgewezen had moeten worden.

(d) (2 punten) In het algemeen is het doel dus een hypothese te vinden zodat uit de 'Hypothese' en de 'Beschrijvingen' van examples in termen van attributen, de 'Classificaties' van de examples *logisch volgen*. We kunnen hierbij op verschillende manieren van 'Achtergrondkennis' gebruik maken. Schrijf één of meer entailment relaties ($\ldots \models \ldots$), waarin je de vier genoemde concepten aan elkaar relateert, die duidelijk maken hoe van achtergrondkennis gebruik wordt gemaakt in algoritmen voor Knowledge-Based Inductive Learning—die bijvoorbeeld in Inductive Logic Programming worden gebruikt.

Antwoord: Er moet natuurlijk altijd gelden dat

$$\text{Hypothese}, \text{Beschrijvingen} \models \text{Classificaties},$$

maar de vraag is op welke manier Achtergrondkennis hieraan wordt toegevoegd. In AIMA sectie 19.2 worden verschillende methoden besproken. In het geval van KBIL geldt

$$\text{Achtergrondkennis}, \text{Hypothese}, \text{Beschrijvingen} \models \text{Classificaties},$$

5. **Prolog**

(a) (4 punten) Beschouw de volgende Prolog code, die het predicaat `d/2` definieert. Voor welke combinaties van invoerwaarden `X` en `Y` is dit predicaat waar? Met andere woorden, wat moet de relatie tussen `X` en `Y` zijn zodat `d(X,Y)` slaagt?

```
d([],[]).
d([X],[X]).
d([X,X|T],Y) :- d([X|T],Y).
d([X,Y|T],[X|Z]) :- X \= Y, d([Y|T],Z).
```

Antwoord: De relatie tussen `X` en `Y` moet zijn dat `X` een lijst met elementen is, en dat `Y` een lijst met dezelfde elementen is, waaruit direct herhaalde voorkomens van elementen zijn verwijderd.

(b) (5 punten) Schrijf Prolog code die het predicaat `range/3` definieert, zodat `range(X,Y,L)` waar is als `X` en `Y` gehele getallen zijn en `L` een lijst is met alle gehele getallen tussen `X` en `Y` (inclusief `X` en `Y`), dus als `L = [X,...,Y]`. Dit moet werken voor *alle* paren gehele getallen `X` en `Y`.

Antwoord:

```
r(X,X,[X]).
r(X,Y,[X|L]) :-
    X < Y,
    X1 is X + 1,
    r(X1,Y,L).
r(X,Y,[X|L]) :-
    X > Y,
    X1 is X - 1,
    r(X1,Y,L).
```

Einde tentamenopgaven