Hertoets Intelligente Systemen (B3IS)

- Het gebruik van boeken, aantekeningen, rekenmachines of andere bronnen is **niet toegestaan**.
- Elke vraag is 9 punten waard, dus je kunt 45 punten verdienen.
- **Antwoord bondig!** Ik trek punten af voor overbodige uitweidingen.

1. **AI, Agenten**
   (a) Russell en Norvig verdelen definities van 'kunstmatige intelligentie' in vier categorieën: Kunstmatig intelligent zijn systemen die *menselijk* of *rationeel*, kunnen *denken* of *handelen*.
      i. (2 punten) Geef een voorbeeld van een systeem dat *rationeel denkt* en een voorbeeld van een systeem dat *rationeel handelt*. Leg uit hoe ze rationeel denken danwel handelen.

      Antwoord: Rationeel denken: een theorem prover, bijvoorbeeld. In het algemeen: een systeem dat "right thinking" implementeert. Het gaat om *correct inferences*.
      Rationeel handelen: een 'rational agent', zoals waar AIMA vol mee staat, een robotstofzuiger bijvoorbeeld. "An agent is just something that acts. (...) A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome." Het gaat hier ook om het 'verstandig handelen' als er geen bewijsbaar optimale handelwijze bestaat.

      ii. (2 punten) Beschrijf een verschil en een overeenkomst tussen deze twee typen systemen. Verklaar beide.

      Antwoord: Dit kan van alles zijn. Een verschil is dat rationeel handelen (acting so as to achieve the best outcome) ook op andere manieren kan dan rationeel te bedenken wat je moet doen. Een overeenkomst is juist dat op in omgevingen waarin rationeel denken over gedrag mogelijk is (statisch, zeker, deterministisch), dit hetzelfde gedrag oplevert als rationeel handelen.

      iii. (1 punt) Om welke van deze vier soorten AI draait het vak *Intelligente Systemen*? Verklaar je antwoord.

      Antwoord: Het draait om rationeel denken, we zijn bezig met het implementeren van (logisch) correcte inferenties.

   (b) (4 punten) Geef een PEAS-beschrijving van een robotstofzuiger.

2. **Logica**
   (a) Sommige inferentie-algoritmen zijn gebaseerd op de resolutieregel.
      i. (1 punt) Geef de resolutieregel. Je mag je beperken tot de propositielogische variant.

      Antwoord: De resolutieregel zegt dat je uit twee disjuncties, waarin complementaire literalen voorkomen, een nieuwe disjunctie mag afleiden (AIMA, p. 252).

      $$\frac{\ell_1 \vee \ldots \vee \ell_k \qquad m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k},$$

      als ℓ_i en m complementair zijn. In het algemeen (AIMA, p. 253) mag m ook een disjunctie met meerdere disjuncten zijn:

      $$\frac{\ell_1 \vee \ldots \vee \ell_k \qquad m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n},$$

      als ℓ_i en m_j complementair zijn. Bijvoorbeeld,

      $$\frac{p \vee q \qquad \neg p \vee r}{q \vee r}$$

      is geldig.

ii. (2 punten) Beargumenteer waarom de resolutieregel *correct* is ('*sound*' in het Engels), dus waarom een conclusie die je ermee kunt trekken uit bepaalde premissen, ook inderdaad het *logisch gevolg* is van die premissen.

Antwoord: Je kunt de geldigheid van de resolutieregel beredeneren met behulp van een gevalsonderscheid. Ik gebruik even het voorbeeld hierboven:

$$\frac{p \vee q \qquad \neg p \vee r}{q \vee r}$$

Bewijs. Allereerst nemen we aan dat de beide premissen $p \vee q$ en $\neg p \vee r$ waar zijn. We maken nu een gevalsonderscheid met betrekking tot de waarheid van de propositie p, die in de ene premisse voorkomt, en in de andere premisse met een negatie ervoor voorkomt.

- Als p waar is, is $\neg p$ onwaar, en moet, vanwege de premisse $\neg p \vee r$, dus r wel waar zijn. Dan is dus ook $q \vee r$ waar.
- Als p onwaar is, dan moet, vanwege de premisse $p \vee q$, de formule q waar zijn, dus ook dan is $q \vee r$ waar.

In beide, elkaar wederzijds uitsluitende gevallen, is $q \vee r$ waar, dus we mogen $q \vee r$ als bewezen beschouwen. QED

(b) (6 punten) Gebruik de resolutiemethode om vast te stellen of de volgende uitspraak waar is.

$$(P(a) \wedge Q(c)), \forall x(P(x) \rightarrow R(x,a)) \models (\exists y R(y,y) \vee \neg \exists z Q(z)).$$

Geef duidelijk antwoord ('waar' of 'niet waar'), en leg ook uit welke stappen je onderneemt om tot je antwoord te komen, en hoe je antwoord volgt uit je methode.

Antwoord: Om te beginnen kun je vrij makkelijk zien dat dit waar is. De eerste premisse zegt onder andere dat $P(a)$ waar is, en de tweede premisse zegt dat voor alle x geldt $P(x) \rightarrow R(x,a)$. Nu is voor het object met de naam a inderdaad $P(a)$ waar, dus is volgens modus ponens ook $R(a,a)$ waar. Dan bestaat er dus een y zodat $R(y,y)$ waar is, zodat natuurlijk ook (voor alle formules A) $(\exists y R(y,y) \vee A)$ waar is, wat (de vorm van) de conclusie is.
Eerst herschrijven we de premissen en *de negatie van* de conclusie naar prenex- en dan naar conjunctieve normaalvorm:

$$\begin{aligned}
P(a) \wedge Q(c) &\equiv P(a), Q(c) \\
\forall x(P(x) \rightarrow R(x,a)) &\equiv \forall x(\neg P(x) \vee R(x,a)) \\
&\equiv (\neg P(x) \vee R(x,a)) \\
\neg(\exists y R(y,y) \vee \neg \exists z Q(z)) &\equiv (\neg \exists y R(y,y) \wedge \neg\neg \exists z Q(z)) \\
&\equiv (\forall y \neg R(y,y) \wedge \exists z Q(z)) \\
&\equiv \exists z(\forall y \neg R(y,y) \wedge Q(z)) \\
&\equiv \exists z \forall y(\neg R(y,y) \wedge Q(z)) \\
&\approx \forall y(\neg R(y,y) \wedge Q(b)) \\
&\approx (\neg R(y,y) \wedge Q(b)) \\
&\approx \neg R(y,y), Q(b)
\end{aligned}$$

We houden dus vijf disjuncties over, waarin alle voorkomende variabelen impliciet universeel gekwantificeerd zijn, en we gaan met de resolutieregel onderzoeken of deze verzameling formules vervulbaar is.

1	$P(a)$	gegeven
2	$Q(c)$	gegeven
3	$(\neg P(x) \vee R(x,a))$	gegeven
4	$\neg R(y,y)$	gegeven
5	$Q(b)$	gegeven
6	$R(a,a)$	resolvent van 1 en 3, met substitutie $\{x/a\}$
7	$\square$	resolvent van 4 en 6, met substitutie $\{y/a\}$

De verzameling van vijf gegeven formules is niet vervulbaar (heeft geen model), dus het is niet mogelijk dat alle premissen en de negatie van de conclusie tegelijkertijd waar zijn. Dat betekent dat een tegenvoorbeeld (wat dat model zou zijn) niet kan bestaan, dus de uitspraak is waar. Je ziet inderdaad, zoals ik intuïtief al beredeneerde, dat de $Q(c)$ in de eerste premisse, en de $Q(b)$ die uit de negatie van de conclusie volgt, geen rol spelen in deze afleiding.

3. **Planning**

(a) (1 punt) Als een state-space search algoritme een plan vindt, hoe ziet zo'n plan er dan uit?

Antwoord: Dit is een lineaire ordening van activiteiten, die een reeks toestanden aan elkaar verbinden, waarvan de eerste de initiële toestand is, en de laatste de doeltoestand.

(b) (1 punt) Noem een voordeel van een partial-order plan boven zo'n plan dat door een state-space search algoritme gevonden wordt.

Antwoord: Het legt niet meer vast dan nodig, 'least commitment'. Er zijn meestal meerdere (vele) lineaire ordeningen consistent met een gegeven partiële ordening, zodat het coderen van een plan als een partiële ordening van activiteiten flexibiliteit geeft die in de uitvoeringsfase van het plan de mogelijkheid geeft van de ene naar de andere lineaire ordening te switchen, afhankelijk van de omstandigheden die zich in de uitvoeringsfase voordoen (voor zover die niet voorzien waren tijdens de planningsfase).

(c) (3 punten) Een partial-order plan (POP) is het resultaat van een zoekproces in een *andere* gerichte graaf dan die van state-space search, namelijk een plan-space graaf. Wat zijn de knopen ('nodes') in deze graaf? Maak een onderscheid tussen (i) de eerste knoop (waar geen kanten ('edges') naartoe lopen), (ii) knopen waar kanten naartoe én vandaan lopen, en (iii) knopen waar alleen kanten naartoe lopen.

Antwoord: Alle knopen corresponderen met een (partial order) plan. De eerste knoop is een leeg plan, met alleen de initiële en de doeltoestand, en niets er tussenin. Het is dus niet uitvoerbaar. Dit geldt voor alle knopen behalve de eindknopen, waar geen kanten vandaan lopen. Alleen de plannen in die laatste knopen zijn compleet, en kunnen uitgevoerd worden.

(d) We willen een schedule maken voor het uitvoeren van de activiteiten in een partial-order plan, waarbij elke activiteit een zekere hoeveelheid tijd kost.

i. (2 punten) (Kijk vast naar vraag ii.) Geef een concrete instantie van zo'n—uitsluitend temporeel—roosteringsprobleem (een POP waarin de activiteiten een zekere duur hebben dus), bepaal hierin het kritieke pad, en leid 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 $s_\ell(a_2) = 15$.

ii. (2 punten) Breid je probleeminstantie van vraag i 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.

4. **Logisch leren**

Stel dat we met het Version Space leeralgoritme een predicaat willen leren dat beschrijft of een object een **rode bal** is. We gebruiken het predicaat *Object*(x, y, z), waar x het formaat is (groot of klein), y de kleur (rood, wit of blauw) en z de vorm (bal, blok of kubus), en we willen dat dit predicaat waar is als een voorbeeld positief is, en onwaar als een voorbeeld negatief is. De voorbeelden zijn natuurlijk gelabeld als positief of negatief. Ons doelpredicaat is een atoom (dus geen complexe formule met connectieven), waarin eventuele variabelen universeel gekwantificeerd zijn. We kunnen dan *generaliseren* door in een predicaat een constante te vervangen door een variabele, en *specialiseren* door het omgekeerde te doen: *Priem*(x) generaliseert *Priem*(17), en *Student*(*ewoud*) specialiseert *Student*(x).

(a) (2 punten) Noem twee *andere* manieren om een (complexe) predicaatlogische formule te generaliseren. (Nota bene: we gaan deze manieren hieronder *niet* gebruiken.)

> **Antwoord:** Je kunt een hypothese in de vorm van een FOL formule ook generaliseren door er een disjunctie van te maken, er andere formules aan toe te voegen die de hypothese ook waar kunnen maken. Bijvoorbeeld: $(\text{bal}(x) \vee \text{rood}(x))$ generaliseert $\text{bal}(x)$. Je kunt ook generaliseren door conjuncten te verwijderen. Zo generaliseert $\text{bal}(x)$ de formule $(\text{bal}(x) \wedge \text{rood}(x))$.

(b) We initialiseren de set G in het Version Space leeralgoritme als het meest algemene predicaat, dus $G_{\text{init}} = \{\textit{Object}(x, y, z)\}$, en we initialiseren de set S als $S_{\text{init}} = \{\}$. We geven het algoritme nu achtereenvolgens vier voorbeelden, waarna het algoritme zal zijn geconvergeerd, want dan geldt $G = S = \{\ldots\}$ (niet leeg). Het is handig om alvast te bedenken op welk predicaat het algoritme zal uitkomen. In dit geval bevatten zowel G als S op geen moment meer dan één atoom.

i. (1 punt) Het eerste voorbeeld is *Object*(*klein*, *rood*, *bal*). Moeten we dit labelen als positief of als negatief? Leg uit.

> **Antwoord:** Het is een positief voorbeeld van een rode bal, namelijk een kleine rode bal.

ii. (2 punten) Welke verzameling (G of S) wordt aangepast, en op welke manier, als gevolg van het verwerken van dit eerste voorbeeld? Verklaar je antwoord. Geef ook de resulterende verzamelingen G en S na deze eerste iteratie.

> **Antwoord:** De verzameling G bevat het predicaat *Object*(x, y, z), dus dat voorspelt correct dat het voorbeeld *Object*(*klein*, *rood*, *bal*) positief is. Het voorspelt namelijk dat *alle* voorbeelden positief zijn. De set S is leeg, en voorspelt dus dat geen enkel voorbeeld positief is. Dat is niet correct, want *Object*(*klein*, *rood*, *bal*) is positief. We voegen dus dat predicaat toe aan de set S, zodat we hebben $S = \{\textit{Object}(\textit{klein}, \textit{rood}, \textit{bal})\}$. De set S is daarmee zo specifiek als hij maar zijn kan, want hij voorspelt nu dat enkel dit ene voorbeeld positief is.

iii. (1 punt) Het tweede voorbeeld is *Object*(*klein*, *blauw*, *bal*). Leg uit of dit voorbeeld een false positive of false negative is, en voor *welk element* van G of S het dat is.

> **Antwoord:** Het is een negatief voorbeeld, dus een false positive. Het predicaat in de set S zegt al dat dit geen positief voorbeeld is, dus daar hoeft niks aan te veranderen. De set G zegt nog steeds dat alle objecten positief zijn, maar dat klopt nu dus niet meer: *Object*(*klein*, *rood*, *bal*) is positief, maar *Object*(*klein*, *blauw*, *bal*) is negatief. Om ervoor te zorgen dat het predicaat in G zo algemeen mogelijk blijft, maar wél deze voorbeelden van

elkaar scheidt, kunnen we het predicaat *Object*(x, y, z) specialiseren, door een variabele in een constante te veranderen. Als we er *Object*$(x,$ *rood*$, z)$ van maken, wordt het eerste voorbeeld *Object*(*klein*, *rood*, *bal*) nog steeds als positief gezien, maar het tweede voorbeeld *Object*(*klein*, *blauw*, *bal*) als negatief.

iv. (1 punt) Welke aanpassing(en) worden er gedaan in de verzamelingen G en/of S, als gevolg van het verwerken van dit tweede voorbeeld?

Antwoord: Het element van G, *Object*(x, y, z) is dus te algemeen, en het moet worden gespecialiseerd, dus een variabele moet door een constante worden vervangen. Het verschil met het eerste (positieve) voorbeeld zit in de kleur, dus we moeten y vervangen door 'rood', want dan wordt het eerste voorbeeld nog wel gecoverd door deze hypothese, maar het tweede voorbeeld niet meer, zoals gewenst. Dus nu is $G = \{$*Object*$(x,$ *rood*$, z)\}$.

v. (2 punten) Na deze twee voorbeelden volgen nog twee voorbeelden, *Object*(*groot, rood, bal*) en *Object*(*groot, rood, kubus*). Beschrijf de aanpassingen die er in de verzamelingen G en/of S worden gedaan bij het verwerken van elk van deze voorbeelden. Geef de verzamelingen G en S na het verwerken van elk van de voorbeelden.

5. **Prolog**

(a) Geef voor elk van de volgende queries het antwoord dat Prolog geeft. Geef ook eventuele substituties als Prolog die doet. Verklaar elk antwoord.

i. (1 punt) `?- tomas = 'Tomas'.`

Antwoord:

```
?- tomas = 'Tomas'.
false.
```

Het zijn twee verschillende atomen, met een kleine en een groter letter.

ii. (1 punt) `?- Tomas = 'Tomas'.`

Antwoord:

```
?- Tomas = 'Tomas'.
Tomas = 'Tomas'.
```

True dus, met de gegeven substitutie.

iii. (1 punt) `?- 2 = '2'.`

Antwoord:

```
?- 2 = '2'.
false.
```

Dit is niet waar, want 2 is een number, en '2' is een constante.

iv. (1 punt) `?- 2 is 1+1.`

Antwoord:

```
?- 2 is 1+1.
true.
```

Want waarom niet? Eerst wordt 1+1 geëvalueerd naar 2.

v. (1 punt) `?- Twee = 1+1.`

Antwoord:

```
?- Twee = 1+1.
Twee = 1+1.
```

True dus, want de complexe term 1+1 wordt toegekend aan de variabele Twee.

vi. (1 punt) `?- Twee is 1+1.`

Antwoord:

```
?- Twee is 1+1.
Twee = 2.
```

(b) (3 punten) Schrijf Prolog code die het predicaat `sorted/1` definieert, zodat `sorted(L)` waar is als `L` een lijst is met gehele getallen die in oplopende volgorde staan.

Antwoord:

```
sorted([]).
sorted([X]).
sorted([X,Y|L]) :- X<Y, sorted([Y|L]).
```

Einde tentamenopgaven