Uitwerking tentamen Kennissystemen 23 juni 1998

1. a) De kennis moet goed gestructureerd zijn (niet alledaagse kennis), het probleemgebied moet beperkt zijn, traditionele methoden voldoen niet, expertise is beschikbaar tijdens het bouwen, menselijke expertise is onvoldoende beschikbaar (duur, snel verouderd, zeldzaam etc.), het is kosten-effectief...
   b) fidelity van een bewering geeft aan in hoeverre die uitspraak waar is, precision geeft aan in hoeverre een uitspraak precies is. De uitspraak "Er zitten veel moleculen in een glas water" scoort hoog wat fidelity betreft (het is waar), maar laag op precision (hoe veel dan?). De uitspraak "De Eiffeltoren is 12,345 cm lang" scoort laag op fidelity (is onwaar) maar hoog op precision.

2. a) 1. probleemidentificatie: hier wordt bepaald wat het probleem is waar een KS voor gebouwd moet worden, wat het systeem moet kunnen, wat de input en output van het systeem is, etc. Tevens wordt een gezamelijk vocabulair gemaakt: de kennisengineer leert de termen die in het domein belangrijk zijn zodat hij/zij met de expert kan praten.
   2. conceptualisatie: door middel van protocol analyse wordt gekeken hoe de expert de taak oplost, wat voor deeltaken er onderscheiden kunnen worden (en wanneer de expert deze uitvoert), en welke kennis er daarbij wordt gebruikt. Er vindt een kennis-niveau analyse plaats.
   3. formalisatie: hier worden de kennistypen, kennisrollen en inferentiesoorten bepaald, en/of het zoekmodel dat gebruikt gaat worden. Hier vindt dus een symboolniveau analyse plaats.
   4. implementatie: het formele model van de vorige fase wordt tot werkend systeem geïmplementeerd.
   b) aspecten: interviewtechniek (ongestructureerd, gstructureerd, protocol analyse, scenario's vergelijken, documenten circuleren), soorten cases (typische, willekeurige, extreme, moeilijke gevallen), protocol omstandigheden (geen of wel tijdslimiet, onvolledige informatie), soorten experts (1 expert, expert + leerling, representatieve groep gebruikers, 2 experts).

3. a) Bayes' regel: $\mathbf{p(y|x) = p(x|y)\frac{p(y)}{p(x)}}$. Met behulp hiervan kan onbekende kennis in bekende vertaald worden. Waar we in geinteresseerd zijn, is de kans op een bepaalde hypothese, gegeven de observaties. Wat we hebben (of kunnen inschatten) is de kans op een observatie gegeven de hypothese, de a priori kans op de hypothese, en de kans op de observatie sowieso (eventueel uitgesplitst naar of de hypothese waar is of niet). Bayes' regel maakt het mogelijk de eerste kans uit te rekenen op basis van de andere. Op deze manier kunnen we telkens de kansen van verschillende hypothesen updaten na het doen van observaties. Het grote voordeel van de methode van Bayes is dat-ie een goede wiskundige basis heeft. Nadelen zijn dat veel getallen nodig zijn, dat de uitkomsten uiteindelijk even betrouwbaar zijn als de (geschatte) a priori kansen waarop ze gebaseerd zijn, en dat de methode van Bayes weinig mogelijkheden voor uitleg biedt.
   b) Keuzes:
   * discrete of continue tijd. Discreet is in stapjes, waarbij er telkens een "volgend" tijdstip is (a la natuurlijke getallen). Bij continue tijd is er tussen elke twee tijdstippen altijd wel weer een tussenliggend tijdstip (a la reele getallen).
   * omvang van kleinste tijdseenheid: alleen van belang bij discrete tijd: hoe groot is het stapje van een tijdstip naar z'n opvolger. Van belang voor fidelity en precision.
   * kwantitatief vs. kwalitatief: bij kwalitatief wordt alleen gerepresenteerd dat een tijdstip voor of na een ander ligt (of dat een tijdsduur langer is dan een ander). Hoeveel dat precies is, of waar een tijdstip precies ligt wordt alleen bij kwantitatieve representatie opgeslagen.

* lineair, parallel of vertakkend: bij linear is de tijd een lijn: de toekomst ligt vast. Bij parallel is dat ook zo, maar zijn er verschillende lijnen voor verschillende objecten: de relatie tussen tijd in de ene lijn en de tijd in de andere lijn is niet eenduidig. Bij vertakkende tijd kan de tijd zich splitsen (als een boom) om verschillende toekomstscenario's mogelijk te maken.
* punten of intervallen: je kunt punten als basiseenheden nemen (met relaties als "ligt-voor" of "komt-na"), maar ook hele intervallen, en relaties hiertussen ("overlapt" bijvoorbeeld).
* discrete of continue waarden: eigenlijk geen keuze van de tijd per se, maar a la discrete of continue tijd maar dan voor waarden van parameters die over de tijd kunnen veranderen.
* vaste waarden op punten of intervallen: heeft een parameter een waarde op 1 punt, of over een interval?

4. a) In de rechtspraak, bij de uitvoering van wetten (uitkering, studiefinanciering...), bij het opstellen van documenten (contracten, testamenten, ...) en bij advies (over procedures of documenten, of over juridische constructies).
   b) Liever forward chaining: bij forward chaining worden alle regels op vuurbaarheid getest, dit levert alleen de laatste op, en die leidt ik_neem_vrijdag_vrij af. Bij backward chaining wordt eerst regel 5 geselecteerd, wat leidt tot nieuw doel het_regent_vrijdag, wat leidt tot nieuw doel het_regent_donderdag, etc, totdat-ie merkt dat het_regent_maandag onwaar is. Pas dan wordt regel 6 geprobeerd, die meteen lukt: veel moeite. Als we een kennisbank hebben die veel conclusies kan afleiden, en we hebben veel initiele data, maar we zijn maar in 1 conclusie geinteresseerd, dan is backward chaining efficiënter, omdat die niet probeert allerlei feiten af te leiden waar we toch niet in zijn geïnteresseerd.

5. a) (1, 0, ?) heeft een onbekende waarde en geen oplossing. De datavector (1, 1, 1) heeft als samengestelde oplossing S_1 & S_2. Voor (0, 0, ?) zijn S_1 en S_2 inconsistent, en dus zijn S_3 en S_4 consistent. Daarvan is S_3 matching.
   b) Oplossingscriteria:
   * conservatief: elk consistente klasse is een oplossing.
   * Positieve coverage: klasse is een oplossing als-ie consistent is, en tenminste 1 observatie "covert".
   * Complete uitleg: een klasse is een oplossing als-ie alle relevante observaties covert.
   * kans drempel: klasse is oplossing als-ie consistent is en z'n a priori kans boven een drempel uitkomt.
   c) MDX is een systeem om leverziektes te diagnostiseren. Het systeem is opgebouwd uit een hierarchie van specialisten en subspecialisten, voor elke abstracte hypothese één. De data wordt gebruikt om een specialist in staat te stellen zijn subspecialisten te (de-) activeren. De methode is "establish en refine": top down stelt een specialist vast of zijn hypothese het geval is. Zo niet stopt-ie, anders roept-ie z'n subspecialisten in om zijn hypothese te verfijnen, gebaseerd op een hierarchisch classificatiemodel.

6. a) * Taal voor specificaties: hierin kunnen de eisen mbt het te configureren object beschreven worden.
   * Kennis over onderdelen: welke onderdelen zijn er, en wat is hun functionaliteit (evt. nog de vereiste onderdelen, vereiste functies, incompatibele onderdelen of alternatieve onderdelen).
   * Kennis over configuraties: beschrijft de beperkingen over mogelijke configuraties (ruimtelijke en temporele beprekingen, beperkingen mbt. de consumptie van resources).
   * Kennis over dubbel gebruik: geeft aan in hoeverre een onderdeel voor meerdere functies gebruikt kan worden (niet, onbeperkt, een aantal keer, serieel of tot een maximale capaciteit).
   b) Het drempel-effect betekent dat een kleine verandering aan de eisen aan het object een grote verandering in de configuratie kan betekenen. Dit gebeurt bij additieve consumptie

van resources: als een resource bijna helemaal gebruikt wordt, en er is toch meer nodig, dan moet er soms een hele nieuwe (grotere) leverancier van die resource komen.
Om het tegen te gaan kun je telkens wat overschattingen maken (dus wat teveel van een resource gaan leveren). Dit werkt niet altijd, soms loop je alsnog tegen een drempel aan. Die overschattingen moeten ook net te grof worden, omdat je door het overschatten soms een optimale oplossing mist.

c) De key-component approach is gebaseerd op de assumptie dat er voor elke belangrijke functionaliteit een (eventueel abstracte) sleutelcomponent is met die functionaliteit. In de initiele configuratie zit elke sleutelcomponent voor de gewenste totale functionaliteit. Door componenten te verfijnen (abstracte componenten door minder abstracte en uiteindelijk concrete componenten vervangen) en recursief required parts toe te voegen (niet-sleutelcomponenten die benodigd zijn door andere componenten) wordt een configuratie bepaald die uiteindelijk nog in een goede "arrangement" moet worden gepast.