Uitwerking Tentamen Kennissystemen

17 augustus 1998

1. a) Deze scheiding bevordert de mogelijkheid van hergebruik: hetzelfde inferentiemechanisme kan met een andere kennisbank gebruikt worden, maar dezelfde kennisbank kan ook met een ander inferentiemechasnisme gebruikt worden.

b) Een representatie is vivid als er voor elk object in de wereld precies 1 symbool is, en als er voor elke simpele relatie in de wereld precies 1 connectie tussen symbolen is. Voorbeeld 1: het stukje werkelijkheid zijn de Amsterdamse universiteiten en de simpele relaties die aangeven of universiteiten samenwerken en of universiteiten concurrent zijn. Neem de representatie met als objecten "VU" en "UvA" en als relaties "Werken_samen(x, y)" en "Zijn_concurrent(x, y)" waarbij precies "Werken_samen(VU, UvA)" en "Zijn_concurrent(VU, UvA)" gelden (en met VU en UvA omgedraaid). Deze representatie is vivid. Neem echter het stukje werkelijkheid dat bestaat uit alle inwonders van Nederland en alle mogelijk telefoonnummers met de relatie die aangeeft dat een bepaalde Nederlander een bepaald telefoonnummer heeft. De representatie bestaat uit alle nationale telefoonboeken van Nederland. Die is niet vivid: er zijn Nederlanders die geen telefoon hebben en dan ook niet in het telefoonboek voorkomen, maar sommige Nederlanders staan er vaker in. Ook staan er bedrijven in: dat zijn geen Nederlanders. Er zijn ook mogelijke telefoonnummers die niet in het telefoonboek staan omdat ze nog niet vergeven zijn. Deze representatie is dus alles behalve vivid.

c) Het gebeurt bij representaties vaak dat er meerdere representaties zijn met dezelfde betekenis (bijvoorbeeld "x + 2" staat voor hetzelfde als "2 + x"). Als je nu voor elke betekenis 1 unieke speciale representatie kunt aangeven, dan heet die representatie een canonieke vorm. Bij polynomen heb je bijvoorbeeld de canonieke vorm waarbij alle zelfde machten van x bij elkaar zijn opgeteld, en de hoogste macht voorop staat, in aflopende volgorde gevolgd door lagere machten, eindigend met de eenheden ($ax^n + bx^{n-1} + ... + cx + d$). Het voordeel is dat je algorithmes kunt ontwerpen die alleen op canonieke vormen werken en vaak efficienter kunnen zijn. Het nadeel is dat het berekenen van een canonieke vorm vaak duur kan zijn (= veel tijd kan kosten).

2. a) klassiek is a la de temporele database: elk object heeft naast de andere attributen ook ruimtelijke coordinaten (2 of 3-dimensionaal). De representatie is dus als lijst objecten met ruimtecoordinaten. Bij nearest-first wordt de ruimte gelijkelijk opgedeeld in regio's. Behalve de objecten met hun gewone attributen en hun ruimtelijke coordinaten worden ook de regio's gerepresenteerd. Per regio wordt bijgehouden welke objecten er in liggen, en wat de buur-regio's zijn. De nearest-first representatie is uitgebreider en dus in principe duurder om bij te houden: bij toevoeging, verwijdering of verplaatsing van een object moet niet alleen dat object zelf veranderd worden, maar ook de informatie van de regio's moet aangepast worden. Het grote voordeel van nearest-first wordt behaald als er vaak queries beantwoord moeten worden van de vorm: "wat is het dichtsbijzijnde object dat aan .... voldoet". Daar is de klassieke representatie niet goed in, en nearest-first is speciaal daarvoor gemaakt. Dus: als dat soort queries relatief vaak voorkomt en veranderingen relatief weinig, gebruik dan nearest-first, anders de klassieke.

b) certainty factors zijn een manier om de mate van onzekerheid van feiten en regels aan te kunnen geven, en worden dus in kennissystemen gebruikt om met onzekere informatie om te kunnen gaan (te representeren en ermee te kunnen redeneren). Certainty factors zijn een getal tussen 1 (totale zekerheid dat iets waar is) en -1 (totale zekerheid dat iets onwaar is). Nul staat voor totale onzekerheid over de waarheid. Gegeven certainty factors van feiten en regels kunnen andere certainty factors uitgerekend worden. Voordeel van de methode is de eenvoud: die maakt het makkelijk te snappen, efficient te implementeren. Bovendien past het uitstekend bij het regel-gebaseerde formaat. Nadeel is dat je veel getallen van de domeinexpert moet verkrijgen (en het is maar de vraag of die ze goed kan schatten), maar erger nog, dat het wiskundig model dat erachter ligt gewoon niet correct is.

3. a) Bij forward chaining wordt begonnen met de initiele data (feiten). Er wordt gekeken welke regels vuurbaar zijn, en daar wordt een selectie uit gemaakt. De conclusies van deze regel(s) worden uitgevoerd, waarna er weer wordt gekeken wat de vuurbare regels zijn, etc. Dus er wordt vanuit de beschikbare data geredeneerd, naar een doel toe. Bij backward chaining begin je met het doel, en wordt er gekeken welke regels dat doel als conclusie hebben. Van deze regels wordt dan de linkerkant het nieuwe doel. Als een doel niet als conclusie van een regel optreedt en ook geen feit is dan kan het opgevraagd worden bij de gebruiker. Als dit niet kan, dan faalt dat doel. Dus hier wordt vanuit het doel terug naar de initiele data geredeneerd. Voorwaarts zou je doen bij bijv. proces controle: op grond van binnenkomende meetgegevens probeer je zo veel mogelijke diagnostische conclusies te trekken (een probleem?). Backward zou je bijvoorbeeld doen als iemand een krediet aanvraagt: je bent alleen geinteresseerd in of die persoon in aanmerking komt voor dat krediet, terwijl je de hoeveelheid vragen die je moet stellen zo beperkt mogelijk wil houden. Bij diagnose kan alletwee gebeuren: eerst voorwaarts om mogelijke diagnoses te bepalen op grond van beschikbare (partiele) informatie. De veelbelovende hypotheses wil je vervolgens testen aan de hand van op te vragen data (en nu gebruik je backward chaining: van de doel-hypothese terug naar de data).

b) het voordeel is dat je een uniforme representatie hebt (die hoef je dus maar 1 keer te snappen, of uit te leggen aan anderen). Dat maakt ook de ontwikkeling van kennissysteem shells nodig. Daar zit al een regel-editor in, en inferentiemechanismes. De keerzijde is dat je allerlei kennis door elkaar heen kunt gooien, waardoor je een grote verzameling regels krijgt zonder structuur. Je bent dus zelf verantwoordelijk voor het behouden van structuur en het scheiden van verschillende soorten kennis.

c) Er kunnen loops in zitten (regels als IF A THEN A), er kunnen inconsistenties optreden (IF A THEN B en IF C THEN $\neg$B als A en C feiten zijn), er kunnen onbereikbare condities zijn (die niet als conclusie van andere regels optreden en ook niet opvraagbaar zijn of als initiele data kunnen optreden), er kunnen niet-bruikbare conclusies zijn (regels waarvan de conclusie geen uitvoer is, maar ook niet als conditie in andere regels optreedt).

4. a) Er is een ruimte A met specificaties (=eisen op het te configureren product) en een ruimte B van partiele configuraties. De toestanden van A zijn specificaties, dwz verzamelingen van eisen op het te maken voorwerp. De overgangen tussen deze specificaties zijn het gevolg van het toevoegen, verwijderen, vervangen of verfijnen van eisen. De toestanden van ruimte B zijn partiele configuraties, dwz niet per se volledige beschrijvingen van het te maken object. Het kan zijn dat bepaalde onderdelen nog abstract zijn, of dat onderdelen nog missen, of dat de ruimtelijke lay-out nog niet volledig is, etc. Overgangen hier zijn het gevolg van het uitbreiden van een partiele configuratie met een nieuw onderdeel, of het verfijnen van een onderdeel (van abstract naar concreet). Tussen deze 2 ruimtes kan heen en weer gegaan worden. Bij een bepaalde begin-specificatie kan een partieel ontwerp gemaakt worden. Als dat echter niet lukt, of als de kosten erg hoog worden, dan kan teruggegaan worden naar de specificatie ruimte. In onderhandeling met de "opdrachtgever/gebruiker" kunnen de eisen aangepast worden, waarna we weer naar de configuratieruimte terug gaan. Dit kan nog enekele keren heen en weer gaan.

b) Een functionele abstractie hierarchie is een boom waarin elke knoop een onderdeel bevat. Zo'n onderdeel kan een abstract onderdeel zijn. Als onderdeel B in de hierarchie onder onderdeel A zit, dan betekent dat dat de functie van een onderdeel A door de functie van een onderdeel B kan worden vervuld. Helemaal onderin bevinden zich de concrete "echte" onderdelen. Het nut hiervan is dat de configuratie hierarchisch kan worden aangepakt: eerst worden de belangrijkste delen geconfigureerd. Hierarchisch kan nu elk deel weer verfijnd worden. Zoals gebruikelijk maakt een hierarchische aanpak een complexe taak doenbaar in redelijke tijd.

c) De methode propose-and-revise (p&r) kan gebruikt worden voor parametrisch design: het te maken object wordt beschreven aan de hand van de waarden van parameters. Aan het configuratiesysteem de taak om de waarden van de parameters vast te stellen. De p&r methode werkt als volgt: er wordt steeds een waarde voorgesteld voor een parameter die nog geen waarde heeft ("propose"). Dan worden de constraints op die parameter bepaald. Als een constraint gebroken is, worden revisies van die parameter voorgesteld ("revise"; dwz een nieuwe waarde). Na de revisie wordt weer een waarde van een volgende parameter voorgesteld.

De drie soorten kennis zijn: kennis om de waarde van een nieuwe parameter voor te stellen (dit kan ook uitrekenen zijn als de waarde volledig bepaald is door eerder voorgestelde paremeters), kennis van constraints tussen paremeterwaarden, en kennis over fixes: wat te doen als een bepaalde constraint gebroken is (en welke fixes zijn in welke gevallen het beste)?

5. a) Als je alleen enkelvoudige fouten toestaat, maak je de assumptie dat maar 1 component van het systeem tegelijk kapot kan gaan (dus een hypothese bestaat uit 1 foute component). Als je meervoudige fouten toestaat, houd je rekening met de mogelijkheid dat er meerdere componenten tegelijk stuk gaan (een hypothese bestaat uit een verzameling foute componenten). Het voordeel van enkelvoudige fouten is dat dat je zoekruimte nog enigszins beperkt houdt. Als je meervoudige fouten toestaat, groeit je ruimte exponentieel. Het nadeel is dat het soms foute antwoorden oplevert: als er in de werkelijkheid meerdere foute componenten zijn. Als er in je domein vaak meervoudige fouten optreden, moet je systeem dat ook aankunnen. Het is een afweging tussen correctheid en efficientie: heuristieken zorgen dat een taak te doen is qua tijd.
b) Aspecten die een rol spelen bij het kiezen van nieuwe observaties zijn de informatie-waarde die ze opleveren (met entropie), de kosten (geld, tijd, ongemak, etc.), de urgentie van mogelijke diagnose (sommige symptomen kunnen op een aanstaande hartaanval duiden, anderen op een verkoudheid), en de coherentie van de dialoog tussen machine en mens.