SISU analys

Nr. 7

KUNSKAPSSYSTEM

av

Carl-Gustaf Jansson

och

Åsa Rudström

Redaktör: Anders Eriksson

SISU - Svenska Institutet för Systemutveckling

Box 1250, 164 28 Kista, 08 - 752 1600, Electrum, Isafjordsgatan 22, Kista

SISU ANALYS nr 7

INNEHÅLLSFÖRTECKNING

INLEDNING

Syftet med detta nummer av **SISU ANALYS** är att ge en inblick i vad som avses med teknologi för kunskapssystem, illustrera användningen av denna, samt att ge vissa prognoser om hur de metoder och tekniker som ingår under denna rubrik kan komma att inlemmas i mer traditionell informationsteknologi.[1]

Artikeln inleds med ett avsnitt om forskningsområdet *Artificiell Intelligens* (AI). Avsnittet tar också upp några tillämpningar av AI-forskningen, och ger en definition av termen *kunskapssystem* (KS), den informationssystemteknologi som utvecklats ur AI-forskningen.

Det andra avsnittet ger ett antal exempel på tillämpningar av teknologin för kunskapssystem. En definition av begreppet *expertsystem* ges, och begreppet illustreras med hjälp av tre exempel på expertsystem: ACE, ISIS och K:BASE. Därefter diskuteras gränssnitt i naturligt språk, med ett exempel: systemet EXPLORER.

Avsnitt tre analyserar vilka egenskaper ett kunskapssystem bör ha och hur dessa kan realiseras. För dem som inte har någon bakgrund inom AI kan detta avsnitt förmodligen upplevas som tungt och teoretiskt. Vi vill därför påpeka att det är fullt möjligt att hoppa över detta avsnitt och gå direkt till avsnittet om hjälpmedel för utveckling av kunskapsbaserade system.

Efter den teoretiska genomgången av kunskapssystem fortsätter artikeln med ett avsnitt om hjälpmedel för utveckling av kunskapssystem. Hjälpmedlen delas upp i författarsystem, generella programmeringsspråk och konstruktionsverktyg, där varje kategori analyseras och exemplifieras.

Som ett komplement till denna beskrivning avslutas artikeln med två avsnitt som avser att ge läsaren en möjlighet att själv fortsätta att studera ämnet. Det första av dessa tar upp vilka universitet, institut och företag som i första hand satsat på forskning inom AI och utveckling av kunskapssystem. Slutligen följer en kort bibliografi, med kommentarer.

---

[1] För en mera utförlig introduktion till kunskapssystem refereras till rapporten *Datorstödda kunskapssystem i framtidens kontor*, av Sture Hägglund, utgiven i TELDOKs rapportserie.

1. ARTIFICIELL INTELLIGENS OCH KUNSKAPSSYSTEM

Artificiell Intelligens (AI) är ett forskningsområde inom dataveten-skapen, där den grundläggande målsättningen är att bygga system, vars egenskaper reflekterar vår uppfattning om tänkandet. Det är dock viktigt att poängtera att forskningen inom artificiell intelligens **inte** strävar efter att simulera mänskligt tänkande i mera total bemärkelse. Vad man gör är att plocka ut delar av vad vi betraktar som intelligent beteende (som till exempel att lösa ett svårt problem genom att dela upp det i mindre, lättare delproblem), och bygga system som efterliknar detta beteende. Om systemet fungerar på samma sätt som det mänskliga beteendet är mindre viktigt, huvudsaken är att systemet kan lösa de uppgifter det är avsett för. Det kan finnas många syften med ett sådant systembyggande:

- skapa system som kan lösa helt nya typer av problem
- göra dagens system mer anpassade till användaren
- genom simuleringar bättre förstå tänkandet.

Det som är mest karakteristiskt för artificiell intelligens som forsk-ningsområde, är den breda ämnesmässiga bakgrund man finner hos de som arbetar inom området. Mer än i något annat delområde av data-vetenskap beaktar AI-forskaren erfarenheter från teoretisk filosofi, lingvistik och kognitiv psykologi, dvs de ämnen som i första hand studerat tänkandet och formaliseringen av tänkandet. Ämnen som neurofysiologi är också av betydelse, även om denna varit mindre.

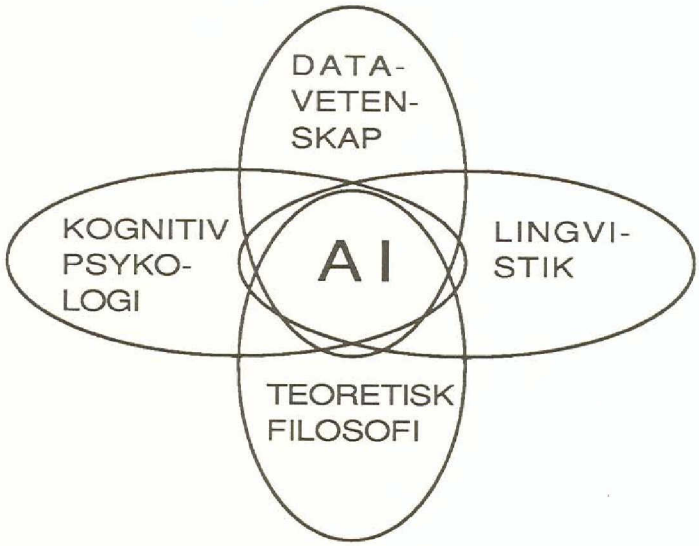


Bild 1: *Artificiell Intelligens är en tvärvetenskap*

En styrka men också en svaghet inom AI är den vetenskapliga meto-den. Det traditionella forskningsresultatet är ett system, samt en åtföl-jande uppsats som stipulerar ett antal önskvärda beteenden hos systemet. Styrkan i en sådan ansats är tvånget att konkretisera idéer i

faktiska system. Svagheten är svårigheten att bevisa att systemet verkligen uppfyller det som hävdas i uppsatsen. Vad man egentligen visar är att systemet uppnår vissa resultat under vissa förutsättningar, och inte att det alltid uppnår önskade resultat. En annan svårighet är hur man bygger vidare på andras resultat, då dessa oftast förekommer just i form av faktiska system, och inte som teorier. Det finns dock en stark tendens inom området att systematisera befintligt kunnande och verka för utvecklingen av en mera sammanhängande teoribildning.

Termen *artificiell intelligens* anses av många som olyckligt vald, och många alternativa termer har föreslagits. Termen myntades av John McCarthy, en av områdets förgrundsgestalter, under mitten av femtiotalet, och har följaktligen en mer än 30-årig tradition i forskningsvärlden. För de som anser att de väsentliga resultaten i ett forskningsområde som AI borde kunna reduceras till rent datalogiska termer följer här ett citat av Allen Newell:

> *"Scientific fields emerge, as the concerns of scientists congeal around various phenomena. Sciences are not defined, they are recognized".*

Ett rimlig ansats är nog att använda begreppet AI när man talar om forskningsområdet, men att välja andra termer när man talar om tillämpningar av de forskningsresultat som framkommit. I denna artikel har vi valt att använda begreppet *kunskapssystem* (KS) när vi talar om en del av den teknologi som utvecklats. En diskussion av begreppet kunskapssystem följer i nästa kapitel.

Det är först under de senaste åren som begreppet AI har spritts i en vidare krets. Detta skall dock inte ses som ett tecken på att AI-forskningen varit fruktlös under 25 års tid, och först under de senaste åren gett några praktiska resultat. Snarare har mängden forskning inom området varit relativt konstant, i första hand koncentrerad till välrenommerade universitet såsom Stanford, Massachusetts Institute of Technology (MIT) och Carnegie-Mellon. De forskningsmässiga framgångarna har också varit tämligen jämnt distribuerade över tiden, även om början av sjuttiotalet respektive början av åttiotalet stått för genombrott i detta avseende. De resultat som nu nått tillämpningsstadiet är de som producerades i början av sjuttiotalet. Vad som hänt under åttiotalet är i första hand att ett antal tillämpningsområden nått en sådan mognad att de blivit kommersiellt intressanta. Detta gäller framförallt de så kallade expersystemen, som beskrivs närmare i nästa kapitel.

En avgörande anledning till det nu kraftigt ökade stödet till både teknologiutveckling och grundforskning är att Japans framtidsbetonade satsning på informationsteknologiområdet kom att välja en ansats som bygger på resultaten från AI-forskningen i början av åttiotalet. Som ringar på vattnet har de flesta större organisationer av global karaktär

valt samma linje, och satsat miljardbelopp på forskning och utveckling inom detta område. Det återstår ännu att se om dessa massiva satsningar under kort tid kan åstadkomma forskningsmässiga genombrott inom ett så komplext område som AI. En mer detaljerad beskrivning av vilka som i dag satsar på forskning inom AI och utveckling inom kunskapssystem följer i ett senare kapitel.

Tillämpningar av AI

Bland de möjliga framtida tillämpningarna av AI-forskningen är det endast följande fem kategorier som nått en sådan mognad att de kunnat lanseras kommersiellt:

- robotteknik
- analys och syntes av bild och tal
- programtransformation
- gränssnitt i naturligt språk
- expertsystem.

Uttryckt i grova termer kan man säga att de två första kategorierna representerar system som realiserar motoriska färdigheter och varseblivning (perception), medan system ur de tre senare kategorierna realiserar mentala (kognitiva) färdigheter.

Forskningen om t.ex robotteknik och bildanalys har alltid varit starkt inspirerad av resultat från teknikområdet och neurofysiologi. Denna typ av system använder sig ofta av analoga och numeriska beräkningar. Forskning om t.ex expertsystem och gränssnitt i naturligt språk har å andra sidan främst hämtat sin inspiration från forskningsområden som teoretisk filosofi, lingvistik och kognitiv psykologi, och domineras därför av symboliska beräkningar, snarare än analoga och numeriska sådana.

På grund av dessa skillnader har de två typerna av system utvecklats mer eller mindre i isolering från varandra, även om arbetet kan ha bedrivits vid samma institution. Det är än så länge ett faktum att dessa metodområden är separata, men vi kommer inom en inte alltför avlägsen framtid att behöva tackla problemet med att integrera dem. Endast ett fåtal forskare, t.ex Marvin Minsky, har varit framsynta nog att inse detta problem på ett mycket tidigt stadium. I Minsky's anda har intresset för icke symboliska beräkningsmodeller (neuronnät) den senaste tiden ökat även för rent kognitiva tillämpningar.

Många anser att de verkligt betydelsefulla teoretiska och praktiska resultaten inom AI-forskningen ligger just på perception och motorikfunktioner. Det är viktigt att framhålla detta eftersom expertsystemtillämpningar i många sammanhang tenderar att dränka övriga tillämpningar. I resten av denna artikel kommer vi dock helt och hållet ägna oss åt att behandla kognitiva system, där representation och beräkningsmodell är av symbolisk karaktär.

Listan ovan tog upp tre kategorier av sådana kognitiva tillämpningar: programtransformation, expertsystem, och gränssnitt i naturligt språk. Expertsystem och gränssnitt i naturligt språk kommer att beskrivas mer utförligt och exemplifieras i nästa kapitel. Programtransformation är ett specialfall av en mer allmän tillämpningskategori vilken som helhet inte är mogen, men troligen kommer att bli av stor betydelse i framtiden, nämligen *stödsystem för systemutveckling*. När det gäller systemutvecklingens sista fas, transformationen av en formell, fullständig programspecifikation till ett effektivt, exekverbart program på en specifik hårdvara/mjukvara, har dock avsevärda framsteg gjorts, och det finns flera kommersiella system som stödjer detta.

Kunskap och kunskapssystem

Termen *kunskapssystem* är en modern synonym till termen *kunskapsbaserat system*. Den senare termen härrör egentligen från en förändring i forskningsinriktning inom AI för ungefär 10-15 år sedan. Dittills hade man framför allt varit inriktad på att bygga generella resonerande system, för att lösa problem inom vilket problemområde som helst. Detta visade sig dock vara mycket svårt, kanske till och med omöjligt, då sättet att lösa ett problem är starkt beroende av vilket problemområde som berörs. Man övergick alltså till att bygga system för att lösa problem inom begränsade problemområden, där systemens problemlösningsförmåga baseras på en djup kunskap om problemområdet. Dessa nya system var alltså *kunskapsbaserade* i högre mån än de tidigare systemen. I dag används dock allt oftare den kortare termen *kunskapssystem*.

Även om det som här angetts är det verkliga skälet till användningen av ordet kunskap är det vanligt att stöta på ett antal rimliga efterkonstruktioner. Alla dessa går ut på att motivera användningen av ordet *kunskap* istället för ord som *data* och *information*. Några exempel på sådana efterkonstruktioner är följande:

- Eftersom AI är starkt påverkat av ämnen som ofta använder ordet kunskap (t.ex teoretisk filosofi) är det naturligt att bibehålla en sådan användning.

- Eftersom de flesta AI-system har en starkt *pragmatisk* karaktär känns användningen av ordet *kunskap* naturlig, på samma sätt som *information* hör ihop med *semantik* och *data* hör ihop med *syntax*.

- Eftersom representationer i kunskapssystem ofta är uttryckta på ett sätt som är *oberoende av användningen* är ordet kunskap lämpligt (enligt en definition som går tillbaka ända till Seneca).

För att avsluta denna frågeställning kan det konstateras att om man vill uppnå överensstämmelse med filosofins kunskapbegrepp, vore begreppet *tros-system* (belief system) lämpligare. Detta begrepp

skymtar allt oftare i forskningsrapporter, men kan knappast förutspås någon framtid i mera kommersiella sammanhang.

Oavsett vilka skäl som kan uppbådas för och emot termen kunskapssystem kommer vi fortsättningsvis att använda denna term för system som uppvisar kognitiva funktioner, och som realiserats med en teknologi som utvecklats inom AI-forskningen. Vi kommer att använda expertsystem och gränssnitt i naturligt språk som exempel på tillämpningar.

2. EXEMPEL PÅ KUNSKAPSSYSTEM

Under de senaste åren har kunskapssystem övergått från att vara prototyper till att bli praktiskt tillämpbara. Vi kommer i detta avsnitt att ge exempel på två typer av sådana tillämpningar: expertsystem och gränssnitt i naturligt språk.

2.1. Expertsystem

Det kanske mest kända tillämpningsområdet för kunskapssystem är expertsystem. Tyvärr råder det här en viss begreppsförvirring. Ofta sätts likhetstecken mellan kunskapssystem och expertsystem, ja till och med mellan artificiell intelligens och expertsystem. Det är dock viktigt att göra skillnad mellan dessa. AI är ett forskningsområde, kunskaps-system en teknologi baserad på detta forskningsområde, medan expert-system är en tillämpning av kunskapssystem.

Ett expertsystem är ett datorstöd som löser problem inom väl avgränsade, om än komplexa problemområden. Det är alltså ett slags intelligent beslutsstöd. För att bedöma om ett visst problemområde är lämpligt, kan man t.ex använda sig av Edward Feigenbaums kriterier:

- Problemet måste vara väsentligt - dvs det är lönsamt med ett system
- En rent algoritmisk lösning är orealistiskt (om någon sådan finns)
- Problemområdet är väl avgränsat
- Tiden för en mänsklig problemlösning är mellan en halv och ett par dagar
- En stegvis utveckling måste vara möjlig
- Ett stort antal typfall bör finnas tillgängliga

Dessutom är det väsentligt att man har tillgång till mänskliga experter, samt att dessa är villiga att dela med sig av sin expertis.

Kompetensen hos systemet ska ligga på ungefär samma nivå som hos en mänsklig expert. Nedan följer några exempel på typiska situationer där ett expertsystem skulle kunna vara till hjälp eller t.o.m. ersätta den mänsklige experten.

- En läkare som ställer diagnos och planerar behandling.
- En jurist som analyserar ett tvistemål eller ett skattetekniskt problem.
- En ekonom som planerar en investering eller gör en komplex revision.
- En kemist som tolkar mätdata.
- En säljare som sammanställer offertunderlag.
- En servicetekniker som gör feldiagnostik.

Klassiska exempel på expertsystem är Mycin (medicin), Dendral (massspektrografi) och Prospector (analys av mineralfyndigheter). Idag finns ett i det närmaste oöverblickbart antal expertsystem inom de mest skiftande områden, såsom justering av soppkokare, felsökning på lokomotiv och signalanalys för stridsledning. Vi tar här upp tre olika exempel. Det första, ACE, är ett diagnossystem för telekablar. Exempel två, ISIS, är ett planeringssystem för en fabrik som tillverkar delar till turbiner, och det sista exemplet, K:Base, är ett mer generellt system, som är speciellt lämpat för finansiella tillämpningar.

Förutom att ACE, ISIS och K:Base behandlar olika problemområden, har de även vissa andra fundamentala olikheter. ACE är baserat på IF/THEN-regler, ISIS på frames och förutsägelser om dessa, medan K:Base framför allt är intressant för sin förmåga att inducera fram sin egen kunskap. Vad som menas med IF/THEN-regler, frames och induktion förklaras närmare i nästa kapitel.

ACE

ACE är ett expertsystem inom telefoni, utvecklat av AT&T Bell Laboratories. Systemet är expert på telekablar och kringutrustning till telekablar, som t.ex kopplingsboxar. Systemet bevakar alla telekablar som hör till en viss telestation, och föreslår förebyggande underhåll likväl som akuta lagningsåtgärder.

För att telestationen ska få reda på var fel har uppstår använder man sig av rapporter om fel, från kunderna och de anställda. För att hantera all denna information hade man även tidigare hjälp av ett datasystem, som sammanställde felrapporter från en databas av klagomål. Dessa sammanställningar studerades sedan av särskilda experter, för att komma fram till var underhållsåtgärder måste sättas in. Problemet för företaget var att sammanställningarna var alltför omfattande för att man skulle hinna ta till vara all information. Därför behövdes ett stödsystem, och ACE byggdes.

Precis som de mänskliga experterna får ACE sin information från det system som sammanställer rapporterna. Det som är speciellt interessant med ACE är att det endast använder denna information, och alltså inte har någon interaktion med sina användare. Detta är en ovanlig egenskap hos ett expertsystem. Användaren kan endast förse ACE med vissa parametrar, t.ex gränsvärden för hur många klagomål som ska komma in innan man bryr sig om att försöka laga ett fel. Förutom detta arbetar ACE helt på egen hand. Det startas en gång per natt, och ringer då själv upp rapportsystemet för att få in de uppgifter som behövs. Saknas uppgifter kan ACE ringa upp igen. ACE gör sedan en bedömning av läget och producerar en lista på var åtgärder behövs och vilka dessa åtgärder bör vara. Listan skickas som ett vanligt brev i kommunikationssystemet hos det operativsystem som ACE arbetar under.

ACE är byggt i OPS4, ett verktyg för utveckling av regelbaserade expertsystem. OPS4 i sin tur körs under Franz Lisp, en lispdialekt, på en VAX-11/780 under operativsystemet Unix. Genom att använda OPS4 har alltså tillgång dels till Lisp och dels till Unix. Tillgängligheten till Unix har möjliggjort kommunikationen med rapportgenereringssystemet, vilket nås via ett Unix-nätverk. Denna integration med övrig programvara är en av ACE's mest attraktiva egenskaper. Just integrationen är annars ett stort problem för många expertsystem. Systemen arbetar ofta på speciell utrustning, som t.ex en lispmaskin, och det är svårt att föra över de data som behövs, och som ofta finns i någon traditionell databas på en annan dator. Sannolikt beror ACE's framgång i detta avseende på att det körs under operativsystemet Unix, som tillåter en hög grad av flexibilitet.

ISIS

ISIS är ett system för planering av driften vid en fabrik som tillverkar delar till ångturbiner. Systemet är utvecklat av en grupp bestående av ett antal personer från Carnegie-Mellon University (CMU), Westinghouse Corp. och Air Force Office of Scientific Research (AFOSR). ISIS används på försök vid en av Westinghouse's fabriker. Systemet gör en plan för hur driften ska gå till, dvs vilka delar som ska tillverkas, på vilka maskiner och vid vilken tidpunkt. En plan uttrycks som ett förlopp av händelser, relaterade till varandra i tiden.

Det stora problemet i en planeringssituation är att ett stort antal olika restriktioner måste vara uppfyllda samtidigt. Ett relativt enkelt sätt att lösa detta problem är att använda sökning. Man börjar med att låta någon restriktion vara uppfylld, och forsätter sedan med nästa, tills alla restriktioner är uppfyllda. Det kan dock ofta hända att uppfyllandet av en restriktion leder till att någon annan restriktion inte kan uppfyllas. Man måste då gå tillbaka, kanske ända från början, och försöka med en ny kombination. Om antalet restriktioner är stort blir det dock omöjligt, såväl för en människa som för en dator, att klara av detta inom rimlig tid. För att kunna lösa problemet måste sökningen begränsas, så att man i varje valsituation bara behöver ta hänsyn till ett mindre antal restriktioner.

En ytterligare svårighet är att olika restriktioner kan vara konfliktelande. Om restriktion A är uppfylld betyder detta att restriktion B inte är uppfylld. Ett exempel på ett sådant fall är om man har en extra snabb men dyr maskin, och en komponent som ska bearbetas av denna maskin eller en annan, långsammare men billigare maskin. Här har vi två restriktioner, priset och tiden. Att använda den billigare maskiner kan leda till att tidsrestriktionen inte kan uppfyllas, medan användning av den snabbare maskinen kan leda till att den färdiga produkten blir för dyr. Här måste det till kunskap om vilken restriktion som är viktigare än den andra, eftersom det är omöjligt att uppfylla båda.

ISIS har försetts med tillräckliga kunskaper för att kunna lösa båda dessa problem. Varje restriktion ges en viss vikt, dvs. ett värde som anger hur viktig restriktionen är. Värdet kan variera från fall till fall, vad gäller en och samma restriktion. Dessa vikter används också för att bestämma var i planeringen man ska börja. I stället för att börja från början startar ISIS med den viktigaste restriktionen, och planerar sedan vidare både framåt och bakåt från denna punkt i planen.

Till skillnad från ACE är användarinteraktionen en mycket viktig komponent för ISIS. Till att börja med får inte ISIS sina data från någon databas, utan direkt från användaren. När alla data är tillgängliga kan ISIS fortsätta arbetet själv, och komma fram med en färdig plan. Viktigare är dock att ISIS kan användas som ett verkligt stödsystem för en mänsklig planerare. Givet vissa förutsättningar kan ISIS föreslå steg i planeringen, visa vilka konsekvenser ett val kan få och tala om när en plan inte kommer att fungera.

En annan viktig egenskap hos ett planeringssystem är att man bör kunna göra ändringar i planen om något skulle gå fel. Avsikten med planen är att den ska exekveras, dvs driften kommer att följa planen och därmed använda en viss maskin för att bearbeta en viss komponent vid ett visst tillfälle. Om en maskin skulle gå sönder faller hela planen, och en ny plan måste skapas från den punkt där man befann sig vid avbrottet. ISIS har förmågan att göra sådana omplaneringar. Detta öppnar dörren för en intressant utvecklingspotential hos systemet. I dagsläget måste användaren tala om för ISIS att något har gått fel. Det finns dock inga hinder för att i en framtid komplettera ISIS med ett automatiskt övervakningssystem, som direkt talar om att planen måste ändras.

ISIS ingår i själva verket i en grupp av expertsystem, Intelligent Management Systems, vilka är under utveckling av CMU, Westinghouse och AFOSR. Detta arbete strävar mot att nå en så stor självständighet som möjligt hos systemen. Att kombinera ett planeringssystem med ett övervakningssystem är typiskt av intresse i detta arbete.

ACE representerar sin kunskap i form av regler. ISIS däremot använder en något mer komplex beskrivning, nämligen frames. Alla objekt i systemet är representerade som frames, såväl maskiner och komponenter som restriktioner på dessa. Systemet är implementerat med hjälp av ett verktyg som var en tidig version av Knowledge Craft, ett verktyg för expertsystemutveckling som säljs av Carnegie Group. Verktyget innehåller ett representationsspråk för frames, baserat på språket SRL (Schema Representation Language), samt en programmeringsmiljö. Dessutom finns en så kallad agendafunktion, som används för att representera de händelseförlopp som beskrivs av planerna.

K:BASE

K:Base skiljer sig från de tidigare beskrivna systemen ACE och ISIS på så sätt att det i själva verket är ett verktyg för expertsystemutveckling. Ursprungligen byggdes systemet som ett stöd för att minimera ränteförluster genom att flera företag samarbetar med sina lånebetalningar. Systemet är dock byggt på ett generellt sätt, där kunskapen om det specifika problemområdet är skild från den generella problemlösningsstrategin. Genom att avlägsna kunskapen om problemområdet återstår ett skelett som kan användas för att utveckla system inom andra problemområden.

Ett stort problem för utvecklingen av expertsystem, generellt sett, är att det ofta är svårt och framför allt tidsödande att samla in och formalisera all den kunskap om problemområdet som systemet behöver för att kunna fungera. Inom problemområden där kunskapen är stabil, som t.ex inom det medicinska området, kan det trots detta löna sig att lägga ner resurser för att få fram denna expertkunskap. Om problemområdet däremot ändras med tiden blir detta orealistiskt - varför lägga ner kanske åtskilliga manår på något som blir föråldrat redan innan det kan tas i drift?

K:Base är utvecklat av Lehman Brothers, Shearson och American Express, och är speciellt anpassat för finansiella tillämpningar. Detta problemområde har just egenskapen att ständigt ändras. En uppsättning regler för att hantera en problemsituation som anses bra en viss dag kanske är helt värdelös en vecka senare. K:Base har därför egenskapen att själv ta fram de regler som behövs för att lösa ett problem inom problemområdet. Det enda utvecklaren av expertsystemet behöver göra är att ange vilka egenskaper det aktuella problemet har, samt hur dessa är relaterade till varandra. Med utgångspunkt från ett antal problem som lösts i dessa termer kan K:Base ta fram regler för hur problemet ska lösas. K:Base är kan alltså själv bygga upp sin kunskap genom att ta fram gemensamma egenskaper för ett antal specifika fall, för att sedan använda denna kunskap vid analys av nya fall.

Låt oss illustrera detta med ett exempel. Vi vill använda K:Base för att skapa ett expertsystem för att lägga upp en aktieportfölj. De egenskaper som ska finnas med är olika uppgifter om kundens ekonomiska ställning, samt hur hur kundens kapital ska fördelas mellan aktier och obligationer. Användaren av K:Base lägger då upp en slags tabell, där man senare kan fylla i värden för de olika attributen. Vissa egenskaper bildar en villkorsdel - i detta fall de egenskaper som rör kundens ekonomiska ställning. Andra egenskaper utgör själva lösningen på problemet - här fördelningen mellan aktier och obligationer. Efter detta är systemet klart att användas. Till en början kan systemet inte ge någon hjälp, men denna situation förändras så snart man börjat mata in ett antal fall - dvs specifika värden för de olika egenskaperna. K:Base skapar nu regler för hur villkors- och lösningsegenskaper hänger ihop. Reglerna har formen av ett så kallat beslutsträd.

När reglerna skapats kommer användaren av aktieportföljssystemet att kunna få hjälp med att lägga upp nya aktieportföljer. Så fort han fyllt i de egenskaper som är markerade som villkor föreslår dessa regler vilken sammansättning kundens aktieportfölj bör ha, dvs. ger värden till lösningsegenskaperna. Det står givetvis användaren fritt att ignorera systemets förslag, eller modifiera dem. I dessa fall kommer systemet att lära sig ytterligare lite mer om aktieportföljer, ungefär som en novis som följer sin läromästare i spåren.

Fördelen med denna typ av system är att man slipper den tidsödande processen med kunskapsinhämtning. Dessutom är själva idén tilltalande, då systemet kommer att bli skickligare och skickligare allteftersom det får en större erfarenhet. En nackdel är förstås att systemet inte har någon större intelligens från början, och dessutom bara kan användas för relativt väl avgränsade problemställningar.

2.2. Gränssnitt i naturligt språk

Att skapa system som förstår naturligt språk är ett eget forskningsområde inom AI. Intresset för detta område har alltid varit stort, vilket är naturligt med tanke på hur ytterligt svårt det är att tänka sig intelligent beteende utan en språklig komponent.

Mycket av språkforskningen utförs av lingvister, men det är numera svårt att dra en skarp gräns mellan den forskning som sker inom "ren" datavetenskap och den som sker inom lingvistik. Det som skiljer är kanske framför allt användningsområdet och skälen till forskningen. Inom lingvistik är man mest intresserad av att analysera språket, varvid AI-teknologi har ett stort användningsområde. Inom datavetenskap är man mest intresserad av att bygga användbara system, och för att ett system ska vara användbart måste man kunna föra en dialog med systemet.

Att använda naturligt språk i gränssnitt till olika typer av system har givetvis många fördelar. Den kanske största fördelen är att man inte behöver lära sig ett speciellt, kanske kryptiskt språk, utan kan använda de språkkunskaper man ändå har. Detta eliminerar också risken för att man ska glömma bort namnet på kommandon etc. Ett gränssnitt i naturligt språk kan också uppfattas som mindre oöverstigligt och skrämmande av en ovan datoranvändare.

Det kan tyckas tveksamt att kategorisera gränssnitt i naturligt språk som kunskapssystem. De flesta av de system som finns idag uppfyller heller inte kraven på ett sådant system. Om man skrapar lite på ytan visar det sig ofta att det som ser ut som naturligt språk i själva verket bara är en utvidgning av t.ex ett frågespråk. Man kan visserligen uttrycka sig på fler, alternativa sätt, men formerna är ändå begränsade.

För att ett gränssnitt i naturligt språk ska kunna sägas vara ett kunskapssystem krävs det mer än så. Systemet måste ha en vokabulär

som dels täcker själva problemområdet, dels omfattar språkets "grammatiska" ord, som hjälpverb, prepositioner osv. En fördel är också om stavningen tillåts vara flexibel, då det är lätt att skriva fel. Om tveksamheter skulle uppstå har systemet alltid möjligheten att fråga användaren vilket ord som avses.

Förutom en heltäckande vokabulär måste systemet ha kunskap om det aktuella språkets syntax. Denna kunskap kan variera i komplexitet, men åtminstone språkets grundläggande syntax måste vara känd. Det är också önskvärt att systemet även kan acceptera och förstå ogrammatiska uttryck. Ofta vill man kunna uttrycka sig på ett mer kortfattat sätt än vad språket egentligen kräver.

Vokabulär och syntax är givetvis centrala för ett gränssnitt i naturligt språk. Men det krävs mer än så. En tidig upptäckt i forskningen om naturligt språk som helhet var att det krävdes oerhört mycket kunskap om världen för att kunna tolka även den enklaste mening. Detta har anförts som skäl till att det är omöjligt att skapa ett system för förståelse av naturligt språk. Om vi däremot talar om gränssnitt till något annat system begränsas detta problem automatiskt. Ett gränsnitt till ett vanligt administrativt system behöver kunskap om anställda, löner, fakturor och liknande, men det behöver inte känna till något om t.ex blommor och växter. I och med att problemområdet begränsats, blir det möjligt för oss att förse gränssnittet med den kunskap det behöver.

Eftersom ett gränssnitt måste kunna fungera i en dialog med sin användare, måste vi ställa ytterligare ett antal krav, typiskt relaterade till dialoger.

- Systemet måste kunna klara av det fenomen som brukar kallas för ellips. Ellips innebär att man i brist på annan information förutsätter att det som tidigare sagts fortfarande gäller.

  Detta förstås bäst genom ett exempel.

  Först ställs frågan "Hur många anställda har avdelning 7?"

  Som nästa fråga vill man då kunna fortsätta direkt med frågan "Och avdelning 9?" utan att behöva upprepa att det är antalet anställda vi är intresserade av.

- Vi vill kunna använda pronomen, t.ex efter frågan "På vilken avdelning arbetar Nilsson?" direkt fortsätta med "När blev han anställd?".

  Systemet måste själv dra slutsatsen att det är Nilsson vi frågar om.

- Systemet måste kunna dra egna slutsatser, och klara av frågor som kanske egentligen består av flera frågor.

  Ett exempel är frågan "Vilka arbetar på Nilssons avdelning?". Här gäller det att först ta reda på vilken avdelning Nilsson hör till, för att sedan fortsätta med frågan vilka övriga som arbetar på samma avdelning. Det kan t.o.m. vara önskvärt att Nilsson inte tas med på den lista som till slut produceras som svar på frågan.

Dessa och många andra egenskaper bör finnas hos ett intelligent gränssnitt i naturligt språk. Tyvärr finns det än så länge inte så många exempel, även om mycket skett under de senaste åren. Vi har här valt att endast ta upp ett exempel.

EXPLORER

Även om namnet Explorer framför allt för tanken till Texas Instruments kraftfulla arbetsstation, så avser vi här ett helt annat system. Explorer är ett gränssnitt i naturligt språk, utvecklat av Cognitive Systems Inc. för ett större oljebolag.

Ett skäl till att vi tar upp just detta system är de omständigheter som gjorde att systemet utvecklades. Oljebolaget har en databas över oljekällor och geologiska formationer i Nordamerika, som framför allt används för att ta fram olika typer av kartor över geografiska områden. För att ta fram dessa kartor används ett kartritningssystem. Tidigare hade detta system ett naturligt språk-liknande gränssnitt mot användarna. Problemet var att användarna inte kunde använda detta gränssnitt. Ett par dagars utbildning ansågs vara tillräckligt, men antingen kunde användarna inte fullfölja utbildningen, eller så glömdes färdigheterna bort, eftersom man kanske inte använde kartsystemet så ofta. Till slut fick oljebolaget anställa särskilda konsulter, vilka arbetade som mänskliga gränssnitt mot kartsystemet. Det var för att undkomma denna situation som Explorer utvecklades.

Explorer fungerar alltså som en expert på användning av det aktuella kartsystemet. Användaren börjar med att mata in en mer eller mindre komplett specifikation av den karta han vill ha. Explorer kompletterar denna specifikation med olika typer av standardvärden, och ger sedan användaren möjligheten att i dialog komma fram till en komplett, skräddarsydd specifikation.

Den ursprungliga specifikationen kan vara mer eller mindre fullständig. Det räcker med att säga något i stil med "ge mig en karta över xxx". Explorer kommer i detta fall att leda användaren steg för steg mot en komplett specifikation, genom att ställa olika frågor. För en van användare kan detta bli onödigt tidsödande. Han kan då i stället ge en komplett specifikation direkt, antingen genom en fullständig mening

eller i form av en slags telegramtext, med förkortningar. Detta är just Explorers styrka - att kunna tolerera olika typer av inmatning, både grammatisk och ogrammatisk.

Explorer tillåter också enkla former av ellips. Efter att ha avslutat en komplett kartspecifikation kan man t.ex be om "En likadan karta över yyy". Explorer kommer då att lägga upp en specifikation med samma värden som den tidigare, över det nya området.

```
PLEASE ENTER A MAP REQUEST:

* Please give me an isopach map from the smackover
* to the eagleford in Hemphill County, Texas, with a
* contour interval of 100 feet and a scle of 1"=2000'

BY SCLE, DO YOU MEAN SCALE (Y OR N)? y
.....
```

Bild 2: *Ett exempel på dialogen med Explorer*

Teorin bakom Explorer härrör från Roger Schank, en av de mest inflytelserika forskarna inom detta forskningsområde. Explorer har kunskap såväl om språket som om problemområdet (dvs om kartor, oljekällor och geografiska formationer). För att tolka en inmatning använder systemet olika typer av kunskap (pragmatisk, semantisk och syntaktisk). Till viss del är tolkningen baserad på förutsägelser om olika företeelser i problemområdet.

Detta sätt att hantera tolkningen av naturligt språk är mycket kraftfullt, men har också vissa nackdelar. Eftersom hela processen framför allt är beroende av kunskapen om problemområdet är det så gott som omöjligt att flytta över ett gränssnitt byggt på detta sätt till ett system inom något annat problemområde. Tiden för utveckling är också relativt lång.

3. EGENSKAPER HOS KUNSKAPSSYSTEM

Vad karaktäriserar ett kunskapssystem? Vi får olika svar på denna fråga beroende på det perspektiv vi väljer att se kunskapssystemet från. Detta avsnitt diskuterar kunskapssystem från ett funktionellt, ett implementeringstekniskt och ett arkitektoniskt perspektiv. I samband med implementeringsperspektivet diskuteras också några aspekter på systemutveckling av kunskapssystem.

3.1. Funktionella egenskaper

Vi kan betrakta ett kunskapssystem från ett rent funktionellt perspektiv, dvs studera systemets egenskaper gentemot användaren. Vad som karaktäriserar ett kunskapssystem är i så fall att systemet bör besitta några av de egenskaper som vi normalt förknippar med intelligent beteende. Som tidigare nämndes är det just denna typ av egenskaper som utgör huvudinriktningen för AI-forskningen som helhet. Några av dessa egenskaper diskuteras nedan.

En viktig del av ett intelligent beteende är att systemet måste kunna kommunicera med användaren på ett intelligent sätt. Att förse systemet med ett användarvänligt och flexibelt gränssnitt är ett steg i denna riktning. Än viktigare är att systemet kan förklara sitt beteende i ter: 'er som är relevanta för användaren, för att denne ska acceptera de slut: atser som systemet drar. Det är också positivt om systemet kan ge konstruktiv feedback och vid behov undervisa användaren.

För att kunna uppfylla dessa krav måste systemet dels kunna resonera om sitt eget beteende, dels resonera i former som påminner om våra egna mentala processer. En förklaring baserad på en helt annan problemlösningsmetod än användarens egen, kommer knappast att vara till någon hjälp.

Ett av skälen till att välja kunskapssystem för att lösa ett problem i stället för mer konventionella lösningar är att problemet inte kan lösas på ett väldefinierat, algoritmiskt sätt. Det är därför av största vikt att systemet kan hantera vaga och osäkra resonemang. Ofta handlar det också om stora problem, där inte all information finns tillgänglig från början. Ett kunskapssystem måste alltså även kunna hantera ofullständig information.

Förklaringsfunktioner och undervisningssystem

Bland de viktigaste egenskaperna hos ett kunskapssystem är förmågan att ge förklaringar, t.ex till varför en viss fråga ställs, varför en viss slutsats dras, eller hur systemet kommit fram till en viss slutsats. Att ge förklaringar ligger också mycket nära att kunna ge konstruktiv kritik på användarens användning av systemet. Om användaren uppvisar brister i sin kompetens bör man dessutom kunna ge undervisning.

Man kan bland existerande kunskapssystem finna exempel på alla dessa egenskaper.

Inom ämnet brukar dessa egenskaper kallas *förklaringsfunktioner, kritiksystem* respektive *intelligenta undervisningssystem*. I allmän bemärkelse rör det sig i alla tre fallen om en överföring av ämneskompetens från systemet till användaren. På sikt borde samma grundläggande metodik kunna tillämpas i alla tre fallen. Svårighetsgraden ökar dock hastigt när man lämnar förklaringar, som här förutsätter en jämförbar kompetens hos system och användare, och försöker förverkliga undervisningssystem. För att kunna ge god undervisning krävs t.ex att systemet har en mycket god modell över användarens kompetens inom problemområdet, samt över hur denna kompetens utvecklas.

Det enklaste sättet att realisera förklaringsfunktioner i kunskapssystem är att föra loggbok över systemets resonerande, och vid behov ge utskrifter ur loggboken. Utskriften görs mer användarvänlig genom att systemets interna representation kompletteras med speciella utskriftstexter. Även om förklaringsfunktioner av denna typ kan ge ett mycket kvalificerat intryck, har de i många avseenden en smak av felsökningssystem i programmeringsmiljöer. Det största problemet är att användarens sätt att lösa ett visst problem inte alltid överensstämmer med det sätt att resonera som förverkligats i systemet, vilket kan leda till att förklaringen enbart upplevs som förvirrande.

Kunskapssystem har ambitionen att representationen av problemet och problemlösningen så nära som möjligt ska ansluta till vår föreställning om våra egna mentala strukturer och vårt eget sätt att resonera. Detta är dock ett svåruppnåeligt mål. Dels vet man inte hur människan resonerar, dels finns det starka individuella variationer mellan olika användare. Detta medför att förverkligandet av förklaringsfunktioner anpassade till olika användare kräver

- en detaljerad analys av hur förklaringar normalt utformas mellan personer av olika kompetens inom ett visst problem område
- en systemarkitektur som tillåter flera olika former av resonemang och framför allt olika detaljeringsgrader
- ett gränssnitt som väljer rätt form av resonemang och producerar utskrifter i en för användaren naturlig form
- en dynamisk modell av användarens kompetensutveckling.

Forskning pågår inom alla dessa områden, med en stark koppling till ren pedagogisk forskning.

Ofullständighet, vaghet och inkonsistens

En viktig funktionell aspekt är också att kunna hantera ofullständig, osäker och vag kunskap. Distinktionen mellan vag och osäker kan upplevas som subtil. Den centrala skillnaden är att vaghet är inneboende i begreppen (vad skiljer t.ex. en kulle från ett berg?), medan osäkerhet normalt tolkas som brist på information. Kunskapssystem angriper ofta problem som har en sådan komplexitet att det är omöjligt att från början ställa upp en komplett kravspecifikation. Systemen utvecklas därför normalt stegvis, vilket innebär att man måste kunna hantera ofullständig kunskap. Systemet måste kunna hantera situationer där fakta är okända, oavgörbara eller står i strid med varandra.

Det traditionella kravet att ett system måste vara fullständigt och som helhet konsistent blir orimligt i dessa fall. Ofullständighet medför ju också att slutsatser som vid en viss tidpunkt betraktats som vedertagna, kan komma att behöva omprövas i ljuset av nya erfarenheter. För att kunna realisera allt detta krävs system som har full kontroll över alla dragna slutsater eller gjorda beräkningar under systemets hela liv, samt över vilka bevis som ligger till grund för varje slutsats eller beräkning (*reasoning maintenance systems*). Resonemang baserade på vaghet och ofullständighet leder oss tillbaka till kravet på förklaringsfunktioner, vilket blir ännu mer påtagligt när systemets dragna slutsatser vilar på denna mångfacetterade grund.

Lärande system

Förklaringar och undervisning är inte bara viktiga i anslutning till den faktiska användningssituationen, utan också i anslutning till utvecklingen av systemet. Som tidigare nämnts utvecklas kunskapssystem normalt stegvis. I och med detta suddas gränsen mellan utvecklings- och användningssituationen ut, på så sätt att systemet utvecklas genom integrerad användning, utvidgning och förfining. I en sådan situation är det av största vikt att systemet har en förmåga till inlärning. Maskininlärning har sedan AI-forskningens början varit ett centralt område, men det är först under de senaste åren som mera konkreta framsteg vunnits. Man bör dock vara mycket försiktig i sina anspråk: det rör sig inte alls om automatisering av en inlärningsförmåga i mänsklig mening, utan om väl preciserade mekanismer vilka syftar till att successivt kunna förbättra ett systems prestanda, vanligtvis i interaktion med systemutvecklaren och användaren. De inlärningsmekanismer som i första hand studerats är följande:

- Att ta fram generella beskrivningar från en stor mängd konkreta företeelser (induktion). Denna typ av mekanism är kanske den hittills mest studerade, och ett flertal system har utvecklats. K:Base, som beskrevs i kapitel två, är ett exempel på ett sådant system.

- Att skapa mer konkreta, operationella beskrivningar från en generell eller ofullständing inmatning. För att klara detta måste systemet ha en omfattande bakgrundskunskap om problem området.

- Att utnyttja analogiresonemang för att överföra kompetens från en problemlösningssituation till en annan. Denna typ av resonemang används mycket ofta i mänsklig problemlösning. Ett exempel är att lära sig atomens struktur genom att jämföra en atom med t.ex solsystemet - givetvis förutsatt att eleven /systemet redan känner till solsystemets struktur.

De flesta traditionella informationsbehandlingssystem saknar de egenskaper som beskrivits ovan. Funktionellt sett avviker därför kunskapssystem markant från traditionella system.

3.2. Implementering

Om vi lämnar det funktionella perspektivet till förmån för ett implementeringsperspektiv får vi andra svar på frågan om vad som karaktäriserar ett kunskapssystem. Den teknologi som används för att bygga kunskapssystem omfattar ett antal programmeringsspråk, programmeringsmiljöer och arkitekturer för hårdvara, som av hävd visat sig kraftfulla och speciellt lämpade för detta ändamål. I allmänhet har dock dessa idéer och produkter redan i hög utsträckning integrerats med mer traditionell informationsbehandlingsteknologi.

Språken Lisp, Prolog och Smalltalk är idag vedertagna generella programmeringsspråk trots att de utvecklats för en AI-forskningsmiljö. Motsvarande programmeringsstilar, dvs. funktionell programmering, logikprogrammering och objektorienterad programmering, liksom de teoribildningar som är kopplade till dessa, utgör i dag delar av datavetenskapens grundvalar.

Även andra viktiga element i teknologin för kunskapssystem har kommit att bli integrerade i den datavetenskapliga repertoiren. Exempel på sådana är formell specifikation av program, programtransformation och automatiserad programmering. Element som mönstermatchning och avancerade kontrollstrategier har också mer och mer integrerats i traditionell metodik.

Detsamma gäller för avancerade programmeringsmiljöer som t.ex Interlisp, och arbetsstationer som Xerox, med den karakteristiska människa-maskin dialogen via grafisk fönsterteknik och musförflyttningar. Genom den stora spridningen av exempelvis Apples Macintoshmaskiner har denna interaktionsteknik blivit var mans egendom.

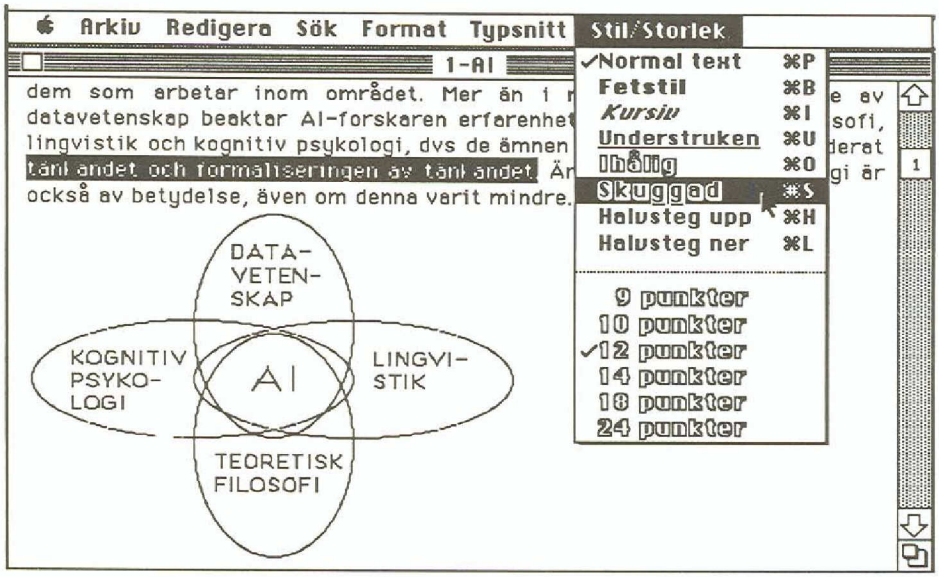


Bild 3: *Ett exempel på interaktion med Macintosh (MacWrite)*

Det är följaktligen uppenbart att det på implementeringsnivå finns en mycket stark överlappning mellan teknologi för kunskapssystem och för datavetenskapen i övrigt. Man bör därför sträva efter att använda en generell datavetenskaplig terminologi på denna nivå, och undvika att profilera tekniker mot ett specifikt område som AI. En AI-forskare kan nöja sig med äran av att många av de inom datavetenskapen dominerande teknikerna har sina rötter i AI-forskningsmiljöer. Marknadskrafterna verkar dock mot denna strävan, eftersom en profilering av en produkt mot ett nytt attraktivt område kan ge marknadsframgångar.

Att utgå från ett antal implementeringsverktyg ur KS-teknologins repertoir garanterar inte att det utvecklade systemet kommer att besitta några av de funktionella egenskaper som angetts ovan. Det förekommer många exempel på felaktiga associationer av detta slag. Det är inte ovanligt med annonser där det hävdas att bruket av LISP som programmeringsspråk garanterar konstruktören ett färdigt system som besitter de mest kraftfulla egenskaper vad beträffar problemlösning, inlärning och utlärning.

Denna diskussion får dock inte tolkas som en nedvärdering av de implementeringsverktyg som används för att bygga kunskapssystem. Som med all teknologi som utvecklats för att tackla svåra problem visar den sig vara mycket konkurrenskraftig även för ett mer allmänt

bruk. Vad som förespråkas är en analytisk ansats som värderar verktygen efter deras verkliga funktion, och inte efter tillhörighet till någon viss forskningsinriktning.

3.3. Systemutveckling

Som tidigare nämnts sker utvecklingen av kunskapssystem normalt stegvis, på ett sätt som nära anknyter till experimentell systemutveckling av traditionella system. De verktyg som diskuterats ovan är också anpassade till denna typ av systemutveckling. Än så länge har intresset varit litet i forskningsvärlden för att formulera systemutvecklingsmetoder, men i och med att industrins intresse för kunskapssystem ökar, ökar också sannolikheten för att sådana metoder kommer att utvecklas inom en snar framtid.

Skillnaden mellan ett traditionellt informationsbehandlingssystem och ett kunskapssystem ligger framför allt i att problemet man vill lösa inte är känt i detalj. Problemet är ofta luddigt formulerat, och det råder osäkerhet både om hur det bör lösas och vilken kunskap som behövs för att kunna lösa det. Något som redan nu har införts är en ny yrkesbeteckning: kunskapsingenjör, dvs en systemutvecklare av kunskapssystem.

Ett grundläggande problem är inhämtandet av kunskapen, som ofta inte finns tillgänglig i formell form. Denna kunskapsinhämtning beskrivs ofta som flaskhalsen i utveckling av kunskapssystem. Kunskapen måste först identifieras, och därefter formaliseras. En grundregel är att områdesexperten bör kunna göra sin analys i första hand på en konkret nivå, och att abstraktioner och formaliseringar överlåts till systemet eller kunskapsingenjören.

Baserat på denna kunskap kan en arkitektur väljas, vilken sedan ligger till grund för en fortsatt stegvis uppbyggnad och förfining av systemet. Vid denna stegvisa uppbyggnad är både förklaringsfunktioner och olika former av maskinell inlärning av största vikt. Ett stort framtida metodiskt problem är hur automatisk och manuell systemuppbyggnad skall kunna integreras.

Det verkar också troligt att återanvändning av gamla system kommer att bli allt viktigare. Detta kan omfatta både att överföra en lösning från ett problemområde till ett annat, att beakta familjer av system för samma problemområde, eller att systematiskt återanvända lösningar i ett och samma system.

3.4. Systemarkitektur

Den mest intressanta och kraftfulla delen av teknologin för kunskapssystem ligger i den *systemarkitektur* genom vilken de eftersträvade funktionella egenskaperna skall kunna uppnås. Detta är samtidigt den

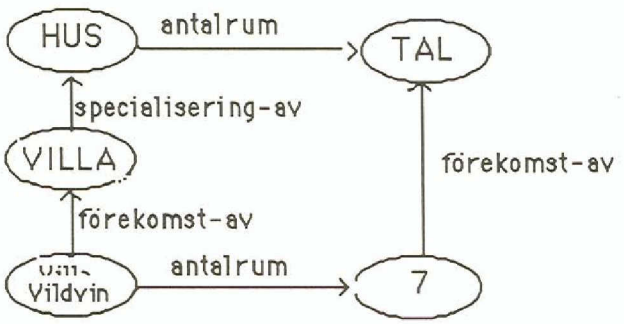
stora utmaningen för fortsatt forskning och utveckling inom AI. Systemarkitekturen omfattar såväl valet av *representation* av den kunskap systemet behöver, som valet av *problemlösningsstrategier*. När man konstruerar ett kunskapssystem är det omöjligt, både i teorin och praktiken, att separera representationen av problemområdet från de problemlösningsstrategier som tillämpas i kunskapssystemet. Systemets prestanda är beroende av en integrerad systemarkitektur, där representation och prolemlösningsstrategier utgör icke separerbara element. Det är meningslöst att analysera en viss form av representation utan att ta hänsyn till dess roll i problemlösningsprocessen. På samma sätt är det meningslöst att diskutera problemlösningsstrategier för ett konkret problem utan att ta hänsyn till vilka representationsformer som skall väljas eller har valts. För en översiktlig beskrivning som denna kan det dock vara lämpligt att framställningsmässigt separera dessa aspekter.

Kunskapsrepresentation

Att välja representationsform är alltid ett centralt problem i informationsbehandlingssystem. Olika metodområden använder olika termer, men den centrala problematiken är densamma oavsett om man talar om *abstrakta datatyper* i programmeringsmetodik, *konceptuella strukturer* i informationsmodellering eller *kunskapsrepresentation* i kunskapssystem.

Det är inte särskilt förvånande att de representationsformer som haft störst betydelse i AI-system ansluter väl till de ämnesområden som starkt påverkat AI-forskningen:

- symbolisk logik, omskrivningsregler (teoretisk filosofi, lingvistik)
- associativa nätverk (kognitiv psykologi, lingvistik)
- mängder, funktioner, relationer, ekvationer (diskret matematik)
- symboler, variabler, poster (datavetenskap)
- neuronnät (neurofysiologi)




```
villa(VillaVildvin)
antalrum(Villavildvin,7)
tal(7)
∀x ( villa(x) -> hus(x) )
∀x ( hus(x) ->
     ∃y (tal(y) & antalrum(x,y) ) )
```

Bild 4: *Ett associativt nätverk och dess motsvarighet i predikatlogik*

Nu är kanske inte själva representationsformen enligt listan ovan det som är mest intressant. Det finns mera allmänna aspekter av representation som är av fundamental betydelse.

Ett exempel på en sådan aspekt är vad ett element i representationen egentligen representerar. Är det en abstrakt idé, som t.ex begreppet "hus"? En konkret företeelse i verkligheten, som t.ex huset jag bor i? Eller är det en företeelse i det symbolbehandlande systemet, t.ex en nod i ett associativt nätverk?

En annan viktig aspekt är vilken status den representerade kunskapen har. Representerar den ett "sant" förhållande i klassisk, logisk mening, vad systemet tror för tillfället, vad någon viss person antar, hoppas eller tror, eller representerar kunskapen vad som anses gälla enligt vissa normer, som t.ex vad som är möjligt eller nödvändigt. En ytterligare fråga i sammanhanget är frågan om hur säker kunskapen är. Är den heltäckande (*definition*), gäller den bara partiellt (*schema*), eller utgör den bara typiska exempel (*prototyp*)? Kan vi ge något kvantitativt eller kvalitativt sannolikhets- eller möjlighetsmått till ett visst kunskapsfragment?

Några ytterligare aspekter är följande:

- Skall kunskapen representeras deklarativt (vad måste gälla?) eller proceduriellt (vad gör man?)?

- Har vi behov av flera beskrivningsnivåer (språk som beskriver språk)?

- Är det möjligt att enas om några grundläggande element och relationer från vilka våra representationer kan byggas upp (semantiska primitiv)?

- Hur ska integrationen av en formell representation och en informell representation i form av text och bildmässiga fragment hanteras?

Det finns uppenbarligen mycket att säga om representation, både generellt och i synnerhet med anknytning till en viss representationsform. Ett gott råd för den som på ett enkelt sätt vill skapa sig en grund för att studera kunskapsrepresentation är att studera den svenska språkläran. Förvånansvärt mycket bygger på elementära språkliga distinktioner.

Representationen i ett visst system bygger normalt på en kombination av de representationsformer som diskuterades ovan. Ett exempel på en vanlig representationsform är *frames*, som har stora likheter med vanliga poster, men även tilhandahåller funktioner som egenskapsarv m.m. Ett annat vanligt exempel är *IF/THEN regler*, som är en

kombination av relationer och omskrivningsregler. Många existerande kunskapssystem bygger sin representation på just frames och if/then regler.

IF	1. the stain of the organism is gramneg 2. the morphology of the organism is rod 3. the aerobicity of the organism is aerobic
THEN	there is strongly suggestive evidence (0.8) that the class of the organism is enterobacteriaceae.

Bild 5: *Ett exempel på en IF/THEN-regel (från expertsystemet Mycin)*

Problemlösningsstrategier

När det gäller problemlösningsstrategier kan man särskilja två olika perspektiv. Enligt den första ansatsen ser man på problemlösning eller resonerande som ett sökproblem, där det gäller att ta sig från ett starttillstånd till ett måltillstånd i en gigantisk sökrymd. Problemet är att finna tillräckligt effektiva regler för att begränsa antalet möjliga vägar genom sökrymden.

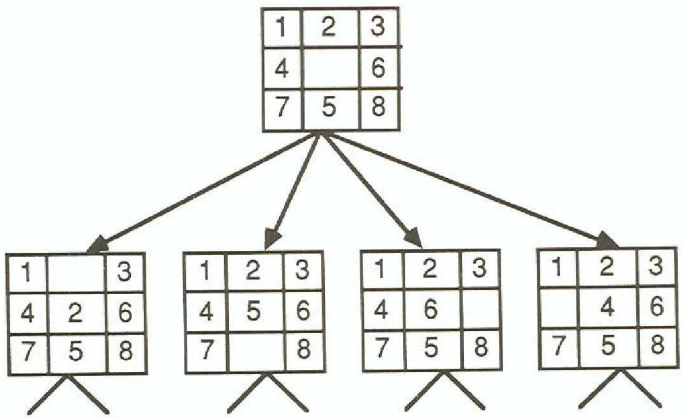


Bild 6: *Problemlösning kan ses som ett sökproblem (åttaspelet)*

Enligt den andra ansatsen bygger problemlösningen på användning av fördefinierade mallar, som beskriver liknande problem eller problemlösningssituationer på en mer generell nivå. När ett nytt problem ska lösas väljs den mall som passar den nya situationen bäst. Informationen om hur problemet ska lösas kan sedan föras över från mallen till det nya problemet.

I själva verket kompletterar dessa två perspektiv varandra. Om utgångspunkten är ett sökproblem, är naturligtvis förekomsten av fördefinierad mallar ett effektivt sätt att begränsa sökvägarna. Är utgångspunkten att tillämpa fördefinierade mallar, kan valet av sådana ses som en sökprocess.

Oberoende av vilket perspektiv man föredrar är följande strategier möjliga:

- studera problemfamiljer, dvs enklare, likartade resp svårare problem av samma slag
- dela upp problemet i så oberoende delproblem som möjligt (*modularisering, divide and conquer*)
- strukturera problemlösningen i ett antal olika abstraktionsnivåer (*stepwise refinement*)
- ställa upp hypoteser och falsifiera (*generate and test, trial and error*)
- ompröva tidigare genomförda resonemang (*backtracking*)
- välja att resonera från problemformuleringen mot målet (*forward chaining, bottom up*) eller försöka reducera målet i termer av problemformuleringen (*backward chaining, top down*)
- tillämpa deduktion (omskrivningar), induktion (generaliseringar), analogier eller abduktion (gissningar, diagnoser)
- applicera sannolikhets- och bevisteori
- utnyttja möjlighet till parallellitet i problemlösningen
- tillämpa mer detaljerade heuristiska regler, som ändå är oberoende av problemområdet, som t.ex. att uppmärksamma symmetrier, gränsfall etc.

Kombinationer av dessa strategier kan vara extra kraftfulla. Ett exempel på detta är att införa hypotetiska delproblem. För den som på ett enkelt sätt vill skapa sig en bakgrund till studiet av problemlösningsstrategier eller resonerande rekommenderas Polyas närmast klassiska bok, "*How to solve it*".

Hur har man då valt perspektiv i några typiska problemlösningsansatser kopplade till konkreta representationer, och i vilken mån har man utnyttjat ovanstående strategier?

I ett *regelbaserat* system betraktas problemlösningen som en sökning. Grundmekanismen är deduktiv, dvs baserad på omskrivningar. Det vanligaste är kanske sökning från start mot mål. Det är också vanligt att dela upp regelsystemet i så oberoende delar som möjligt, medan det är mindre vanligt att införa flera abstraktionsnivåer, eller uttnyttja möjligheter till parallellitet.

I ett *framebaserat* system bygger problemlösningen på att man redan har ett scenario för liknande problem. Lösningsstrategin går ut på att

finna ett tillämpligt scenario, för att sedan fylla i detta med det som gäller för det aktuella problemet. Såväl uppdelning i delproblem som uppdelning i olika abstraktionsnivåer är vanlig. För att finna ett scenario används någon form av mönstermatchning, möjligen baserad på analogi.

3.5. Betydelsen av olika aspekter

Förhoppningsvis har ovanstående karakteristik av *kunskaps-representation* och *problemlösning* fyllt syftet att illustrera att arkitekturen hos ett kunskapssystem är något mycket centralt men också komplext.

Vi vill hävda att en meningsfull användning av begreppet kunskapssystem bör ha som krav att ett system ska uppfylla vad som sagts ovan för alla dessa aspekter för att få kallas kunskapssystem. Detta innebär att ett kunskapssystem bör uppfylla de funktionella kraven, vara implementerat m.h.a verktyg av angivet slag samt ha en arkitektur som tillgodoser valet av kunskapsrepresentation och problemlösningsstrategi enligt ovan. Därmed skulle begreppet kunskapssystem fungera som en bra varudeklaration.

Nu är den praktiska verkligheten en annan. Begreppet kunskapssystem har använts och kommer att användas för system där endast någon av dessa aspekter är beaktad. Vi kommer troligen alltid att behöva uppleva att ett godtyckligt system implementerat i Prolog eller Lisp kallas för kunskapssystem, hur beklagligt detta än må anses. De flesta som seriöst arbetar inom området delar nog uppfattningen att system som inte beaktar de funktionella kraven inte heller bör betecknas som kunskapssystem. Behovet av en arktitekonisk aspekt är nog mera omdiskuterat, då det finns många exempel på framgångsrika system vilka implementerats direkt i språk som Lisp eller Prolog utan något ytterligare metodiskt stöd.

Vår uppfattning är att arkitektur för kunskapssystem har samma betydelse som arkitektur för byggnadsverk. Mycket små, väl avgränsade och enkla projekt ("kojor"), kan i princip vem som helst bygga, utan arkitektonisk skolning. Enskilda personer eller grupper av personer med exceptionell förmåga kan till och med klara av stora komplexa projekt ("slott"), utan denna skolning. I många stycken speglar kanske dessa båda ytterlighetsfall den faktiska situationen ifråga om kunskaps-system. Ett mindre antal omfattande och ambitiösa system har byggts av kompetenta grupper. Ett mycket stort antal små prototyper har också byggts utan något uppenbart behov av arkitektoniska betraktelser.

Erfarenheten från andra områden visar dock att om komplexa projekt skall kunna genomföras regelmässigt av grupper med normal förmåga, så är kraven på genomtänkt metodik stora. Arkitektonisk erfarenhet är en förutsättning för att utveckla en sådan metodik.

Ett skäl till varför vissa utvecklingsprojekt har negligerat den arkitektoniska diskussionen har redan nämnts. De system som byggs är så små och enkla att det verkar onödigt att "skjuta mygg med kanoner." Ett annat skäl ligger i relationen mellan implementeringsverktyg och arkitektur. Ovan har vi i presentationsmässigt syfte infört en klar gräns mellan dessa aspekter. Nu är verkligheten inte så enkel. De representationsformer och beräkningsmodeller som hjälpmedlen för implementering tillhandahåller är oftast enkla varianter av de representationsformer och problemlösningsstrategier som beskrivits ovan. I den mån implementeringshjälpmedlen förbättras kan de bättre och bättre ansluta till de mest grundläggande arkitekturerna. Ett sådant lyckosamt språk är Prolog, som vid sin tillkomst var en i det närmaste optimal kompromiss mellan en god men enkel arkitektur, och en praktiskt realiserbar implementering.

Att ett sådant språk är ett kraftfullt verktyg för vissa typer av problem bör dock inte förleda oss att anse en arkitektonisk metodik obehövlig.

4. HJÄLPMEDEL FÖR UTVECKLING AV KUNSKAPSSYSTEM

Hjälpmedel för utveckling av kunskapssystem kan delas in i följande tre kategorier:

- generella programmeringsspråk och interaktiva programutvecklingsmiljöer
- generella författarstöd
- författarstöd avsedda för system inom ett visst problemområde.

De två första kategorierna kommer att närmare beskrivas i följande två avsnitt. Den sista kategorien kommer med tiden att bli allt viktigare eftersom effektiv konstruktion av kunskapsbaserade system med nödvändighet måste göras mer orienterad mot varje problemområde, om tillräckligt kraftfulla hjälpmedel skall kunna tillhandahållas. För närvarande existerar dock endast ett fåtal sådana system, ex. inom problemområden som organisk kemi, diagnostik av kretskort och investeringsanalys.

Detta kapitel avlutas med några ord om den utrustning på vilka de beskrivna hjälpmedlen finns tillgängliga samt några avslutande råd vid val av hjälpmedel.

4.1 Generella programmeringsspråk

Generella programmeringsspråk ger i huvudsak stöd på implementerings nivå. De språk som dominerat implementeringen av KS har varit funktionella språk, logikprogrammeringsspråk och objektorienterade språk. Det är genom korsbefruktningar av dessa språkliga inriktningar som den fortsatta språkutvecklingen sker. En speciell kategori är de språk som i hög grad bygger på användning av produktionsregler.

Generellt kan man säga att alla programmeringsspråk som traditionellt använts för konstruktion av KS besitter de flesta av följande egenskaper:

- enkla beräkningsmodeller med väldefinierad formell semantik
- likformig representation av program och data, dvs. en indirekt möjlighet att hantera program som om de vore data
- rekursiva och symboliska datastrukturer
- deklarativa och rekursiva programbeskrivningar
- flexibla kontrollstrukturer
- funktioner för mönstermatchning
- interaktiv och inkrementell programutveckling
- rika programmeringsmiljöer.

En typ av system som ännu inte kommit att användas i praktiken är system för *Reasoning maintenance* (eller synonymt *Data Dependency Maintenance).* Denna typ av system kan dock förväntas bli mycket betydelsefulla i framtiden. Grundidén är att på ett systematiskt sätt logga varje beräkning och slutsats som görs under systemets exekvering, tillsammans med indata, utdata, beräkningsregler och tidpunkt. En sådan systematisk loggning ger en teoretisk möjlighet att gå tillbaka till en godtycklig punkt i beräkningen. Ett exempel på ett sådant system är WATSON, utvecklat av James Goodwin i Linköping.

Innan vi går över till en beskrivning av de nämnda programmeringsstilarna bör det påpekas att den kommersiella tillämpningen av KS-tekniken inneburit anpassningar och närmanden till konventionella programmeringsspråk såsom Pascal, Fortan, C och Ada. Å ena sidan kan utvecklingsverktyg implementeras direkt i dessa språk. Å andra sidan kan utvecklingen ske i den ursprungliga, ofta Lispbaserade miljön, medan det färdiga systemet överföres till en exekveringsmiljö byggd på något av de mer konventionella språken. Hjälpmedel för att automatisera denna överföring (på fackspråk kallad portering) börjar nu finnas tillgängliga. Exempel på ett system som översätter ett regelbaserat system till ett program i C är TRC(Translate rules into C) utvecklat vid North Dakota University. Hjälpmedel för portering planeras också för verktyget EPITOOL.

Funktionella språk

Det traditionella programmeringsspråket för implementering av kunskapssystem är *Lisp* . Lisp, som utvecklades på slutet av 50-talet av John McCarthy, är det näst äldsta programmeringsspråk som fortfarande är i allmänt bruk (efter Fortran). Den språkmässiga kärnan i Lisp har tillsammans med språk som APL inspirerat utvecklingen av en kategori av språk som brukar kallas *funktionella språk.* En förklaring till namnet är att alla beräkningar beskrivs i form av tillämpning av ett stort antal enkla funktioner. De mer renodlade språken i denna kategori såsom ML (Meta Language), har ännu ej fått någon större spridning. Ett undantag är SCHEME, detta förmodligen tack vare dess nära släktskap med LISP och en överlägset bra lärobok i programmeringsmetodik som refereras i ett appendix.

Lisp däremot har fortfarande en stark ställning mycket på grund av sin roll som bakgrundsspråk i många andra verktyg och författarstöd. Det är en förutsättning för varje seriös användare av KS-teknologi att i viss utsträckning behärska Lisp, eftersom detta språk så länge utgjort ett andra modersmål inom AI, och därmed präglar AI-litteraturen som helhet.

Trots att språket Lisp är mycket enkelt, kompletteras språkkärnan normalt av rika men svåröverskådliga programmeringsmiljöer, t.ex InterLisp (Xerox), MacLisp (DEC20/Tops20), ZetaLisp (Symbolics) och FranzLisp (VAX/UNIX).

Logikprogrammeringsspråk

På samma sätt som funktionella språk utgör logikprogrammeringsspråk en stor grupp, men med ett tydligt fokus: Prolog. Prolog utvecklades visserligen i London och i Marseille i början av 70-talet, men utgör egentligen en i det närmaste perfekt kompromiss av språkliga experiment vid amerikanska AI-laboratorier, baserade på en önskan att kombinera teorembevisning med ett effektivt programmeringssystem. Idealbilden av logikprogrammering är att kunna specificera ett program rent deklarativt, i termer av en förenklad form av symbolisk logik. För alla problem av intresse krävs det dock i praktiken att konstruktören känner till språkets relativt komplexa, om än renodlade kontrollstrategi.

Prolog är ett oöverträffat verktyg för att göra små prototyper till kunskapssystem. Språkets koncisa, deklarativa uttrycksformer gör att Prologprogram i många stycken är läsbarhetsmässigt överlägsna de system som tas fram med hjälp av de verktyg som kommer att beskrivas nedan. Prolog lider fortfarande brist på tillräckligt bra programmeringsmiljöer, samt får för flera implementeringar, prestandamässiga problem vid större tillämpningar.

Logikprogrammering är ett mycket aktivt forskningsområde (*Japans Fifth Generation Computer project* är bara ett expempel), och arbetet syftar till att ta fram logikprogrammeringssystem som bättre uppfyller kraven på

- deklarativ programbeskrivning för stora, realistiska system
- bättre prestanda (logikprogrammeringsspråk har stora potentialer att utnyttja parallell exekvering)
- bättre programmeringsmiljöer.

Objektorienterade språk

Objektorienterade programmeringsspråk är en grupp som haft lika stor betydelse som de tidigare två grupperna, men saknar det naturliga fokus som Lisp respektive Prolog utgör i dessa.

Grundidén i ett objektorienterat språk är att strukturera programmet kring beskrivningar av klasser av objekt. Såväl deklarativa som proceduriella delar av programmet är knutna till dessa objektklasser. Det är också mycket vanligt att organisera klassbeskrivningarna efter generalitet i en vanligtvis hierarkisk struktur. Egenskaper hos en mera generell klass av objekt ärvs av de mer specifika klasserna. Denna egenskapsärvning gäller både för data och proceduriella beskrivningar och möjliggör en hög grad av beräkningsmässig ekonomi.

Det klassiska objektorienterade programmeringsspråket är Simula, som utvecklades i Oslo i slutet av 60-talet. Språket Smalltalk, som föddes i Xerox Parc i början av 70-talet, har haft en mycket långsam utvecklings- och spridningstakt, men den officiella versionen av språket från 1980 utgör en ur programmeringsmiljö synpunkt modernare variant av Simula.

Varken Simula eller Smalltalk har i någon nämnvärd grad används för utveckling av kunskapssystem. Dock har dessa språk inspirerat till utvecklingen av objektorienterade utvidgningar till språk som Lisp och Prolog, och har därigenom kommit till stor användning för KS-konstruktion. Exempel på sådan utvidningar är

- för Lisp: Flavors (utvidgning av ZetaLisp) och LOOPS (utvidgning av InterLisp)
- för Prolog: Vulcan och Mandala (utvidgningar av Concurrent Prolog), samt SIMPSOS.

Regelbaserade språk

Historiskt sett har användningen av IF/THEN regler dominerat vid konstruktion av expertsystem. Detta mycket tack vare de många framgångsrika systemen framtagna vid Heuristic Programming Project vid Stanford University, teoretiska arbeten av Nils Nilsson samt de modeller för mänsklig problemlösning som framförts av Simon och Newell vid Carnegie-Mellon University. Medan man i Stanford primärt arbetade med regelbaserade skal (se nedan) av typ EMYCIN, valde man vid Carnegie-Mellon att utveckla regelsystem som utvidgningar till Lispsystem. Detta har lett fram till den sk. OPS-familjen, vars nu aktuella medlemmar är OPS5 och OPS83. Det sistnämnda språket är en konvertering av OPS5 till Pascalmiljö.

Regelbaserade språk har vid praktisk användning stora överens-stämmelser med logikprogrameringsspråk, även om de ur ett teoretiskt perspektiv bygger på skilda ansatser. Exempel på regelbase-rade/logikprogrammerings språk är PLANNER (MIT), ROSIE (Rand Corporation) och MRS (Stanford).

Hybridverktyg eller hybridspråk

Hybridverktyg har fått sitt namn av möjligheten att med hjälp av verktyget fritt kunna komponera representationsformer och problem-lösningsstrategier. Ett hybrid verktyg eller hybridspråk har normalt någon av de ovan beskrivna språktyperna som grund.

Exempel på forskningsverktyg som inspirerat utvecklingen av denna typ av verktyg är AGE och RRL. Både AGE och RRL implementerades i InterLisp vid HPP i Stanford 1979-80, och tillhandahåller verktygslådor från vilka konstruktören kan komponera sin egen systemarkitektur.

Hittills kommersiellt tillgängliga hybrid verktyg kan indelas i

- stora verktyg som KEE (Intellicorps), ART (Inference Corporation), Knowledge Craft (Carnegie Group) och EPITOOL (Epitec). Det marknadsledande verktyget är KEE. Verktygen är ofta implementerade i Lisp på arbetsstationer av typ Xerox 1186, Explorer eller Symbolics.

- medelstora/små hybridverktyg som Goldworks, NexpertObject, Personal Consultant och Acorn.

4.2 Författarstöd

Med författarstöd avses datorstöd som ger en reell hjälp vid kunskapsinhämtning, systemdesign, uttestning samt fortlöpande underhåll och utveckling. Sådana system innehåller normalt inlärnings- och förklaringsfunktioner. Vi kan säga att denna typ av verktyg ger stöd på en funktionell nivå. De författarstöd som hittills utvecklats ger följande former av stöd:

- hjälp med utformning av enkla förklaringsfunktioner och med utformning av människa/dator-dialogen
- tillhandahåller ett kraftfullt specifikationsspråk
- konsistenskontroll av modellen
- effektivare människa/dator dialog i själva författarstödet
- stöd för förfining av kunskapsbasen
- stöd för inledande begreppsanalys
- undervisning av konstruktören

De flesta författarstöd är alla mer eller mindre inspirerade av John McCarthys *Advice taking system* som presenterades på 50-talet. Grundidén är att konstruktören kan uttrycka sig i en delvis informell, ofullständig form och att systemet formaliserar, fyller i luckor, förklarar och konkretiserar.

Skal

Den enklaste typen av författarstöd är de hjälpmedel som brukar kallas *skal*. Ett skal stödjer konstruktion av sytem med en fix arkitektur, dvs. en viss kombination av representation och problemlösningsstrategi. Namnet härrör från idén att ta ett befintligt fungerande system och gröpa ur alla dess ämneskunskap tills endast strukturen återstår - en analogi till snäckans skal. Det klassiska exemplet på ett skal är EMYCIN som är en urgröpning av expertsystemet MYCIN. MYCIN är ett diagnossystem för infektionssjukdomar. Fördelen med skal är att de starka restriktioner som läggs på konstruktören kan möjliggöra mer kraftfulla hjälpfunktioner, t.ex. när det gäller utformning av förklaringar och en detaljerad dialog.

Enligt en nyutkommen översikt indelas skal som producerar regelbaserade system i:

- stora skal av typ IBM's ESE, som körs under VM och MVS, till en kostnad på i snitt 25-50 K$
- medelstora skal av typ Guru på IBM PC och Nexpert på Macintosh, till en kostnad på i snitt 5 K$
- små skal av typ APES, till en kostnad av 1 K$.

Kraftfulla specifikationsspråk

Inom AI området har en flora av deklarativt orienterade specifikationsspråk utvecklats vilka bygger på en kombination av associativa nätverk, första ordningens predikatlogik samt klassisk mängdlära och funktionsteori. Dessa språk uppvisar uttrycksmässigt stora likheter med specifikationsspråk inom ex. informa- tionsmodellering och programspecifikation. Skillnaden är att inom AI området, specifikationerna i princip är exekverbara. Exempel på sådana är ORBITS (utvecklat vid MIT), KRL (utvecklat vid XEROX Parc) och KLONE (utvecklat vid Bolt Beranek and Newman). Ett modernt språk av denna kategori är KRYPTON som utvecklades 1983 vid Fairchild. Ibland kan det vara svårt att skilja ett sk. specifikationsspråk från den typ av hybridverktyg som beskrivits ovan. Jämför man dock exempelvis KRYPTON med KEE ser man snart klara olikheter i karaktär, vilket rättfärdigar den gjorda distinktionen. Medan KEE snarast är en komplex programmeringsmiljö på objektorienterad och regelbaserad grund, utgör KRYPTON ett genomtänkt specifikationsspråk med klar semantik.

Konsistenskontroll av modeller

Det klassiska exemplet på ett författarstöd är TEIRESIAS som utvecklades 1977 av Randall Davis vid HPP i Stanford. TEIRESIAS stödjer utveckling av EMYCIN-baserade system och tillhandahåller enkla former av alla nämnda stödfunktioner. Även när det gäller konsistenskontroller är TEIRESIAS ett föregångssystem. Generellt kan man påstå att konsistenskontroller ännu är ett aktivt forskningsområde. Vad som hittills kan åstadkommas är snarast syntaktiska kontroller och utvecklade varianter av felsökningsfunktioner i programmeringsmiljöer.

Det är mycket troligt att effektiva stöd för konsistenskontroll inte kommer att vara generella utan domänspecifika, dvs. att stödsystem kommer att utvecklas för väl avgränsade problemklasser, såsom medicin, kemi eller ekonomi. Det finns redan flera exempel på sådana system, t.ex. regelkontrollfunktionen i expertsystemet ONCOCIN, och systemet CONCHE som gör konsistenskontoller vid utveckling av system inom organisk kemi.

Förenklade författardialoger

De flesta hjälpmedel, såväl författarstöd och programmeringsmiljöer som konstruktionsverktyg, använder den kraftfulla grafiska interaktionsteknik som bygger på grafik, fönsterhantering, menyer och pekdon. Denna teknik är har sina rötter i en AI-miljö (Allan Kays arbete vid Xerox Parc), men finns nu på de flesta arbetsstationer, såväl avsedda för KS som andra tillämpningar. Samma teknik finns nu allt ofta också på persondatorer.

Ytterligare förbättringar av dialogen kan naturligtvis göras genom att utnyttja färg och andra medier. Någon större spridning av sådana tekniker är ännu ej aktuell i samband med hjälpmedel för KS-konstruktion. Ett undantag är integration av KS i CAD system.

Vidare kan fördelar uppnås genom en dialog i stiliserat språk eller i bildform. Fördelarna med att använda ett stiliserat språk är framför allt att läsbarheten ökar, samt att man slipper kraven på alltför exakta format.

ROSIE är ett regelbaserat språk, utvecklat 1981 vid Rand Corporation, i vilket alla specifikationer sker i naturlig men stiliserad form. LESK, utvecklat vid SRI, har samma målsättning. I såväl ROSIE som LESK är syftet främst att öka läsbarheten. I KLAUS, ett mera ambitiöst system, även detta utvecklat vid SRI, har man också haft ambitionen att lindra kraven på exakthet i inmatningen.

Förfining av modeller

En viktig stödfunktion i TEIRESIAS är att underlätta specifikationen av nya regler, genom att använda tidigare definierade regler och generaliseringar av dessa. I SEEK som utvecklades 1984 vid Rutgers University har man byggt vidare på denna idé. Detta system stödjer konstruktören i den systematiska generaliseringen och specialiseringen av befintliga regelmängder.

Induktiva verktyg utgår från stora mängder av konkreta fallbeskrivningar, och genererar generella beskrivningar som täcker fallen. Det är viktigt att först etablera en mängd av väldefinierade attribut i vilka alla exempel uttrycks. Normalt måste dessa attribut vara fixerade. De induktiva systemen etablerar relationer mellan olika attribut och attributvärden och representerar dessa antingen i form av beskrivningar av klasser av objekt, eller mer generellt som regler vilka anger samband mellan olika objekts attribut. I vissa fall byggs begrepps- eller regelhierarkier upp. Systemen producerar ofta sina resultat i form av beslutsträd. De flesta induktionsystem bygger på ett fåtal centrala algoritmer (ID3 resp. AQ).

Bland de kommersiellt tillgängliga verktygen märks:

- stora induktiva verktyg som EX-TRAN, RuleMaster och TIMM till en kostnad av ungefär 20 K$.
- små induktiva verktyg som Expert-Ease, ASSISTANT86 och Super Expert till en kostnad av ungefär 5 K$.

Expert-Ease (1983), Super-Expert, och EX-TRAN är alla utvecklade av ITL (Intelligent Terminal Ltd), som är specialiserade på denna typ av verktyg och har starka anknytningar till den fortsatta forskningen inom induktionsområdet, vilken bedrivs bl.a. vid Turinginstitutet i Glasgow.

De större verktygen har en bättre koppling till annan programvara och underlättar integrationen av induktivt formulerade och manuellt definierade beskrivningar. Detta är skillnaden bl.a. mellan EX-TRAN och Expert-Ease. Ofta producerar de mindre verktygen ett beslutsträd som kan användas för klassificering. Större verktyg stödjer konvertering mellan beslutsträd och mer konventionell regelform, samt användning av den senare formen i ett regelbaserat system. Ett exempel på ett integrerat system är ADVICE, ett forskningsverktyg utvecklat 1983 vid University of Illinois. ADVICE är egentligen ett hybridverktyg (se ovan), men tillhandahåller induktiva komponenter, t.ex. CLUSTER, som skapar en hierarki av klassbeskrivningar från en oordnad mängd objektbeskrivningar. Verktyget är implementerat i Pascal för VAX/UNIX.

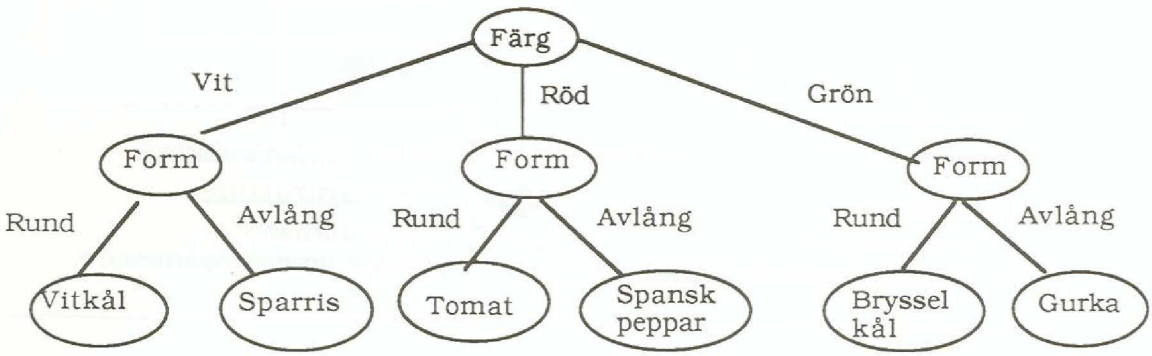


BILD 7: *Exempel på ett beslutsträd. Egenskaper utgör noder och värden på egenskaper utgör grenar. Löven i trädet utgör utfall vid klassifikation.*

Induktiva verktyg utgör dock ej någon ersättning för manuell förfining, annat än för mycket speciella tillämpningar av klassifikationskaraktär. Effektiv integration av manuell respektive induktiv förfining av kunskapsbasen är fortfarande ett relativt outforskat område. Svårigheterna att bedöma relevansen av den formaliserade kunskapen snarare ökar än minskar. Det står dock helt klart att induktiva verktyg kan utgöra ett gott stöd i utvecklingsarbetet även om manuell förfining av kunskapsbasen länge kommer att dominera.

Stöd för den inledande begreppsanalysen

Inget av de ovan nämnda hjälpmedlen ger något stöd för den inledande begreppsanalys som är en förutsättning vid kunskapsinhämtning inom ett nytt problemområde. Tvärtom är de starkt beroende av en relativt fullständig kartläggning av för problemet relevanta företeelser och egenskaper.

Det finns två typer av hjälpmedel som kan stödja denna inledande fas av kunskapsinhämtningen:

- protokollanalyssystem som exempelvis bygger på *hypertext* och/eller *mindmap* -begreppen
- Intervjustödsystem som systematiskt försöker generera frågor vilkagör relevanta faktorer och dessas samband explicita.

En *hypertext* är en samling textfragment, bilder etc vilken är strukturerad genom länkar mellan de individuella fragmenten. I detta fall kan vi tänka oss att hypertexten utgöres av de fragment av dokument och intervju protokoll som ligger till grund för analysen. Markeringar av relevanta ord och stycken resp. bildelement kan överlagras och dessa element kan sedan i sin tur sammanbindas med länkar.

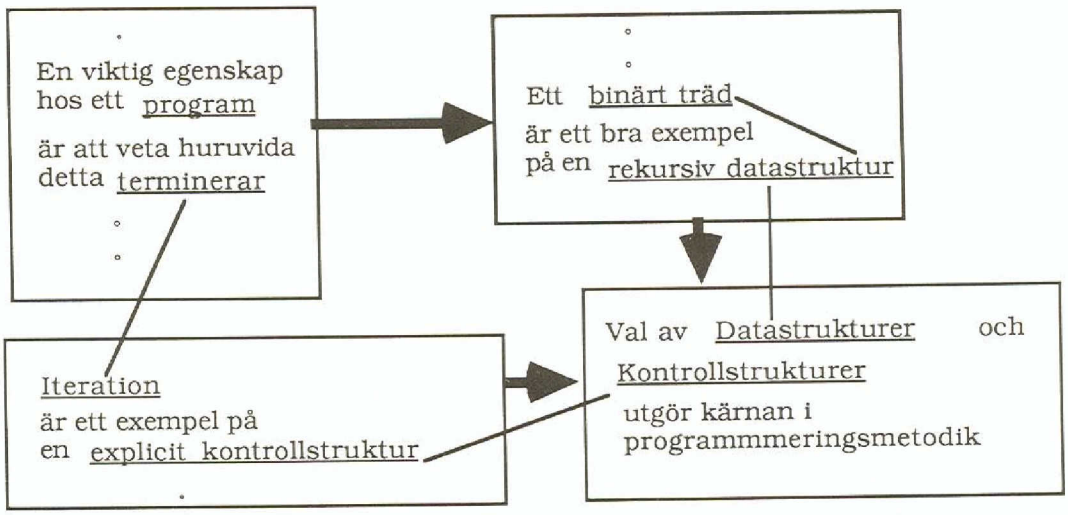


Bild 8: *Exempel ur ett protokollanalyssystem.*

En *mindmap* eller minneskarta är ett informellt nätverk av för en viss problemställning relevanta begrepp och dessas samband. Det är uppenbart att den överlagrade strukturen av markeringar och länkar som beskrevs ovan utgör en datorbaserad minneskarta för det aktuella problemområdet.

Ett exempel på ett hjälpmedel som innehåller här skissade funktioner är KADS, ett stödsystem för kunskapsinhämtning som tagits fram inom ramen för det europeiska forskningsprogrammet ESPRIT.

Ett exempel på den andra typen av system är ETS (Expert Transfer System) utvecklat1984 vid Boeing i Seattle. ETS stödjer formulering av IF/THEN regler genom riktade, datorgenererade intervjuer, baserade

på en psykoterapeutisk metod kallad *Personal Construct Theory* utvecklad av George Kelly på 50-talet. Svagheten med Kellys metod i detta sammanhang är att de bäst lämpar sig för klassifikationstillämpningar. Resultat av analysen bokföres succesivt i matrisform, en sk. *Repertory Grid*. ETS har använts för ett stort antal interna projekt inom Boeing. En trevlig egenskap hos ETS att ett regelsystem kan genereras för ett visst typ av regelbaserat verktyg, som t.ex EMYCIN, OPS5 eller Prolog.

	Datalinjen	Datateknik linjen	Systemvetenskapliga linjen
Inslag av matematik	1	4	2
Inslag av samhällsvetenskap	2	1	4
Inslag av teknik	2	4	2
Lång utbildning?	2	4.5	3
Bra grund för forskarkarriär	2	5	3
Studie intensitet	3	4	3

Bild 9: *Exempel på ett* ***Repertory grid****, där sambanden graderas på en skala från 1 till 5.*

Ett annat exempel är ROGET, ett författarsystem för diagnossystem utvecklat vid HPP i Stanford 1983. Roget genererar EMYCIN-regler från datorgenererade intervjuer.

4.3 Utrustning

De hjälpmedel som beskrivits ovan finns tillgängliga för någon eller några av följande typer av utrustning:

- större tidsdelade datorer, ex VAX/ UNIX, IBM maskiner under VM och MVS
- arbetsstationer speciellt inriktade mot KS-verksamhet, som t.ex LMI/LAMBDA, SYMBOLICS, XEROX 1186 och EXPLORER
- konventionella arbetsstationer av typ SUN, APOLLO
- persondatorer av typ IBM PC, IBM AT och MACINTOSH

De mot KS-verksamhet inriktade arbetsstationerna brukar ofta gå under benämningen *Lispmaskiner*. Detta begrepp används ofta i en vid bemärkelse trots att det egentligen har en specifik innebörd: en Lispmaskin är en maskin vars maskininstruktioner, alternativt mikrokod, är speciellt anpassade till exekvering av Lispsystem. Alternativt kan man tänka sig en mer konventionell maskin där operativsystemet är skrivet i LISP. Med samma innebörd kan man också tala om Prologmaskiner och Smalltalkmaskiner.

Generellt kan man säga att behovet av kraftfulla, dedikerade arbetsdatorer för AI-tillämpningar har överskattats under en övergångsperiod. När dessa maskiner introducerades i början av 80-talet medförde de en drastisk prestandaökning, som snabbt gjorde ett stort antal tillämpningar praktiskt realiserbara. Vid stora krav på beräkningskapacitet utgör de fortfarande en förutsättning, men med tanke på att många KS-tillämpningar är relativt begränsade, samt att persondatorer blivit alltmer kraftfulla, kommer denna senare kategori av maskiner att växa i betydelse som maskinvara för KS-tillämpningar.

4.3 Val av hjälpmedel

Visserligen är det mycket svårt att göra några välgrundade jämförelser, men några ord på vägen vid val av hjälpmedel kan dock ges.

Är man i den lyckliga positionen att hitta ett författarstöd för sitt speciella problemområde så bör naturligtvis ett sådant väljas. Detta är dock få förunnat. Även de flesta andra av de generella författarstöd som beskrivits i avsnitt 4.2 befinner sig ännu på forskningsstadiet och har mycket begränsad tillgänglighet och tillämpbarhet.

Vad som återstår är då ett *skal* som åtminstone stödjer en snabb utveckling av enkla systemskisser inkluderande standardiserad användardialog och förklaringsfunktioner. Exempel på ett sådant skal är det Prolog baserade APES. En lovande utvecklingslinje är integrationen av skal och 4 GL verktyg så som exemplifieras av GURU. En rekommendation till den som har vaga begrepp om omfattningen av det tänkta systemet är att inledningsvis välja ett system av ovan nämnda slag för att till låg kostnad kunna utveckla en allra första prototyp. Normalt finns dessa skal tillgängliga på förhållandevis billiga persondatorer som Apple Macintosh och IBM PC. Troligtvis kan kostnaden för både utrustning och programvara av detta slag ligga under 30 000 SEK.

I de fall enkla lösningar visar sig otillräckliga är naturligtvis en hybridmiljö att föredra, där godtyckliga kontroll- och datastrukturer kan väljas för att skräddarsy systemets arkitektur. Har man tillgång till erfarna programmerare är en programmeringsmiljö à la COMMONLISP med objektorienterade och logik- programmerings påbyggnader att föredra. Mindre vana implentatörer kan dock föredra ett integrerat verktyg som KEE och EPITOOL. Det förtjänar dock att upprepas att i bägge dessa fall ligger priset inklusive hårdvara i bästa fall en faktor tio högre, än i de system som rekommenderades för de inledande systemskisserna, dvs 300.000- 1.000.000 SEK.

En klar trend visar dock att allt fler kraftfulla hybridverktyg göres tillgängliga på mer konventionella arbetsstationer som ex SUN. Ett bra verktyg på en konventionell arbetsstation av typ SUN kan idag kosta omkring 150.000 SEK. Förhoppningsvis kommer bra utvecklingsmiljöer att mycket snabbt också vara tillgängliga för den senaste genera-

tionen av persondatorer såsom IBM PS2-60 eller Macintosh II, vilket kommer att ge tillgång till en effektiv hård/basprogramvara för mellan 75.000 och 100.000 SEK.

Även i ljuset av detta senare alternativ kan det dock rekommenderas dem som vill pröva på KBS tekniken att för inledande systemskisser använda billig programvara för enkla persondatorer, för att vid en eventuell senare reell satsning välja någon av de dyrare alternativen. Alltför många har redan förköpt sig på dyr utrustning och programvara för att i realiteten lösa ganska blygsamma problem.

5. FORSKNING OCH UTVECKLING

Som nämndes inledningsvis dominerades den tidiga AI-forskningen av de tre amerikanska universiteten Stanford, MIT och Carnegie-Mellon. Alla dessa universitets AI-laboratorier har haft en stor bredd inom området, med företrädare såsom John McCarthy, Edward Feigenbaum, Marvin Minsky och Allen Newell. I Europa är det bara universitetet i Edinburgh som haft någon längre tradition inom området.

Visserligen har dessa universitet bibehållit sin starka ställning, men situationen idag är en helt annan. Ett stort antal universitet runt om i världen producerar forskningsresultat av hög kvalitet. Idag bedrivs framgångsrik AI-forskning av amerikanska universitet som Rutgers, Yale, Columbia och Illinois i Urbana Champaigne, samt av europeiska som Bryssel, London, Paris och Ljubljana.

I Sverige bedrivs forskning vid många högskolor, framför allt vid institutioner för datavetenskap men också för lingvistik, filosofi och matematik. De högskolor som satsat extra mycket på AI är Linköping, Uppsala och Stockholm.

Institutionen för datavetenskap i Linköping har under Erik Sandewalls ledning byggt upp en omfattande utbildnings- och forskningsverksamhet inom AI-området med forskningslaboratorier för bildanalys, analys och syntes av naturligt språk, kunskapsrepresentation, utvecklingsmetodik och programmeringsmetodik.

I Stockholm är verksamheten uppdelad på tre institutioner. Vid institutionen för Data- och Systemvetenskap bedrivs utbildning och forskning inom AI (forskningslaboratoriet SYSLAB), med betoning på kunskapsrepresentation, inlärning och utvecklingsmetodik. Vid institutionen för Numerisk Analys och Datalogi arbetar man med bildanalys. Slutligen arbetar institutionen för Datorsystem med arkitekturer för logikbaserade språk.

I Uppsala domineras verksamheten av UPMAIL (Uppsala Programming Methodology and Artificial Intelligence Laboratory). UPMAIL har länge varit internationellt erkända för sina insatser inom logikprogrammering. Man har bland annat mycket goda kontakter med det japanska FGCS-projektet. Utöver detta bedriver man också annan forskning och utbildning inom KS.

Många nybildade institut och forskningsavdelningar på stora företag konkurrerar framgångsrikt med universiteten. En rik flora av företag har vuxit upp, som på olika sätt försöker tillämpa och kommersialisera de mest lovande forskningsresultaten. Avsikten med detta avsnitt är att försöka bringa reda i denna något oöverblickbara situation.

Om vi börjar med de amerikanska institut och företag som först satsade på AI forskning så bör åtminstone Xerox, SRI (Stanford Research Institute), Rand och BBN (Bolt, Beranek and Newman) nämnas. Alla dessa har varit aktiva inom området sedan början av 70-talet, och karakteristiskt för dem alla är att satsningen på AI framförallt varit en investering i kompetensutveckling, samt att resultaten mestadels har grundforskningskaraktär.

Vid Xerox Parc i Palo Alto har Xerox byggt upp ett forskningscenter som i kompetens och resurser överglänser de flesta universitet. Xerox Parc's avdelningar för informationsteknologi har sin tonvikt på programmeringsmiljöer, kunskapsrepresentation, undervisningssystem och en naturlig dragning mot kontorstillämpningar. SRI, som är ett institut i Menlo Parc nära Stanford University, har en framgångsrik AI-avdelning, med en bred orientering mot kunskapsrepresentation, teorembevisning, gränssnitt för naturligt språk och expertsystem. Nils Nilsson har under flera år varit vetenskaplig ledare vid SRI. Rand Corporation i Santa Monica har en tonvikt på militära och juridiska tillämpningar. BBN är ett företag beläget i Massachusetts i Boston. BBNs avdelning för informationsteknologi har sin styrka i kunskapsrepresentation, undervisningssystem, kognitiva modeller och kvalitativa modeller.

Ett flertal stora amerikanska företag har under slutet av 70-talet och början av 80-talet följt dessa pionjärer i spåren, och satsar nu starkt på KS teknologi. Exempel på sådana företag är IBM, Digital, Unisys, Hewlett-Packard, Texas, Bell-labs, General Electric och Boeing. Flera av dessa har gjort satsningar av storleksordningen 20-30 miljoner dollar. Alla kombinerar en stor andel långsiktig kompetensutveckling med konkreta tillämpningar, vilka har krav på mer kortsiktig lönsamhet. Vanligt är också att man integrerar KS-teknologi i sina övriga produkter.

Det första större företag som satsade på KS-teknologi med ett medvetet kommersiellt syfte var Schlumberger, ett företag som bl.a. bedriver oljeprospektering. Det konkreta problem som deras utvecklingsavdelning i Connecticut skulle lösa var analysen av lutningen hos geologiska lager (The Dipmeter Advisor). Schlumberger hävdar att deras satsning varit ekonomiskt bärkraftig och har nu, förutom sin ursprungliga avdelning, starka intressen i Fairchilds AI Laboratory och i BBN.

I skuggan av dessa jättar har ett stort antal mindre företag dykt upp. Dessa kan delas upp i följande kategorier

- utveckling av designverktyg
  (Carnegie Group, Inference Corporation, Teknowledge, Intellicorp)
- utveckling av generella gränssnitt för naturligt språk
  (Cognitive systems)
- agent för andras produkter i kombination med konsultverksamhet
- utveckling av applikationer för specifika problemområden, t.ex finans
  (Synintelligence)
- utveckling av grundprogramvara, t.ex. Lisp- och Prologsystem
  (Gold Hill, Quintus)
- utveckling av arbetsstationer inriktade mot generella AI-tillämpningar
  (Symbolics, LMI)

Den ökade graden av kommersiell mognad hos KS-teknologin bestyrks av de ökade satsningar som IBM nu redovisar. Nu görs satsningar såväl på vetenskapliga centra (Paris och Palo Alto), forskningscentra (Yorktown Heights) och på utvecklingslaboratorier (Nordiska laboratoriet). Satsningarna omfattar såväl grundprogramvara som utvecklingsverktyg och tillämpningar. Vid Yorktown Heights har man bl.a. utvecklat ett Lisp-system och ett ofta refererat expertsystem (YES/VMS). I Paris ligger tonvikten på Prolog, kunskapsrepresentation och naturligt språk. I Sverige driver Nordiska Laboratoriet ett projekt kring gränssnitt för naturligt språk. IBM driver även i Sverige mer kommersiellt inriktad konsultverksamhet (Kista).

I Sverige kan de fåtaliga företag som ännu satsat på KS-teknologi klassificeras på samma sätt som de redovisade amerikanska företagen.

Stora företag som Ericsson, ASEA, Philips, Televerket och Ellemtel upprättar utvecklingsavdelningar, huvudsakligen i syfte att utveckla en egen kompetens för framtiden men också i syfte att genomföra mer kortsiktigt lönsamma projekt. Televerket har dock kanaliserat mycket av sina satsningar inom detta område via företagen Telelogic och Infologics.

Företag av typ INFOLOGICS marknadsför andra företags produkter och ger utbildning och konsulthjälp i anslutning till dessa produkter. EPITEC i Linköping är det enda svenska företag som satsat på utveckling av ett hybridverktyg (Epitool) av den typ som beskrivits ovan. Epitool är i första hand avsett för större arbetsstioner. ZYX utvecklar och marknadsför Prolog-system, bl.a. för UNIX-system och för Macintosh. ZYX har nyligen i samarbete med Ångpanneföreningen bildat ett företag som skall bedriva konsultverksamhet med KBS

teknik. Detsamma gäller också Cap Gemini som håller på att bygga upp en grupp med denna inriktning.

NOVACAST marknadsför kunskapssystem inriktade mot metallurgi och gjutningsprocesser, samt induktiva verktyg från ITS (Intelligent Terminal Systems). ITS drivs av Donald Michie som är en tongivande gestalt inom europeisk AI-forskning. Michie är också vetenskaplig ledare för Turinginstitutet i Glasgow, ett forskningsinstitut som är ledande inom området maskininlärning, med betoning på induktiva metoder.

Ett flertal nationella och multinationella program ger sitt stöd både till grundforskning, tillämpad forskning och utveckling. Det numera nästan klassiska exemplet är Japans FGCS (Fifth Generation Computer Systems) från 1982. Detta program är en framtidssatsning som omfattar KS-teknologi, arkitekturer för parallella processorer och en ny generation av integrerade kretsar. FGCS organiseras av Institute for New Generation Computer Technology (ICOT) i Tokyo, men en stor del av arbetet genomföres vid industrier som Fujitsu, Hitachi och Mitsubishi. Det som uppmärksammats mest i FGCS är kanske valet av Prolog som en grundval för den fortsatta utvecklingen. Ett antal varianter av Prologmaskiner är också hittills det konkreta resultatet av programmet.

Oberoende av den eventuella framgången hos den mycket ambitiösa japanska satsningen, så har denna fått en oerhörd effekt för den mängd resurser som satsas på kunskapssystem runt om i världen. Av dessa skall först nämnas MCC och DARPA's Strategic Defense Program. MCC (The Microelectronics and Computer Technology Corporation) är en organisation bildad av ett stort antal amerikanska företag, bl.a. Boeing, Control Data, Unisys, Lockhead och Digital. MCC har en årsbudget på 65 miljoner dollar, och flera av dess huvudområden omfattar KS-teknologi.

DARPA's Strategic Defense Program är en kraftigt utökad satsning (1 miljard dollar på 10 år) på informationsteknologi för den nationella säkerheten. De projekt man prioriterat (förarlösa fordon och stridsledningsstöd) medför en stark fokusering av AI-området. Detta projekt har i sin tur inspirerat ett europeiskt samarbetsprojekt inom ramen för Eureka-programmet. Projektet kallas Prometeus, och har till syfte att studera användningen av KS-teknologi vid utveckling av en framtidsbil. Flera svenska företag, universitet och institut är inblandade i detta arbete.

De största satsningarna i Europa sker annars inom ramen för EG's program ESPRIT. ESPRIT har en 5-års budget (1984-88) på ungefär 1.5 miljarder dollar och stödjer konkreta samarbetsprojekt mellan industrier, universitet och statliga myndigheter. Ett fåtal amerikanska bolag med europeiska dotterbolag som IBM, ITT och Digital deltar också i samarbetet. Det rör sig i första hand om tillämpad forskning och

informationsöverföring till industrin. Många ESPRIT-projekt kombinerar KS-teknologi, informationsmodellering och databasmetodik. Den rena grundforskningen inom AI har fått ett mera begränsat stöd i form av COST-13 programmet, som i första hand syftar till att befrämja kommunikationen i det europeiska AI-forskarsamhället.

I England har ESPRIT-programmet kompletterats med Alvey-programmet, med en liknande målsättning som ESPRIT och med en budget på 500 milj dollar under 5 år. Tyskland och Frankrike har också forskningsprogram av samma typ. När det gäller Östeuropa är situationen mera oklar, men både Ungern (Institute for Computer Coordination) och Jugoslavien (Jozef Stefan Institute) har visat intresse för att delta i ESPRIT. De har dock ännu ej antagits.

Den svenska satsningen tar sig uttryck bl.a. i form av de ramprogram för informationsbehandling och datavetenskap som stöds av Styrelsen för Teknisk utveckling (STU). De ovan nämnda universitetsinstitutionerna med sina forskningslaboratorier får i första hand stöd via ramprogrammen. Medel tillkommer även lokalt via universiteten (ex. IIT programmet i Linköping). Under de senaste åren har också två kollektivforskningsinstitut bildats, SISU och SICS (Swedish institute for Computer Science). Båda dessa finns på Elektrum i Kista. Till skillnad från SISU som stöds av en ganska omfattande grupp företag av olika storleksordning, stöds SICS av en liten grupp större företag t.ex Ericsson, IBM och ASEA. Båda instituten har särskilda avdelningar för kunskapssystem utveckling.

Förutom det arbete som bedrivs vid industrier, universitet och institut har även andra större organisationer visat sitt intresse. Exempel på sådana organisationer är Televerket och Statskontoret. Det senare har ett projekt kallat *Växthusprojektet*, vars syfte är att utvärdera möjliga tillämpningar av kunskapssystem inom statsförvaltningen.

Vill man följa utvecklingen kommer det idag ut en oöverskådlig flod av böcker, tidningar och proceedings från konferenser. Några referenser till litteratur ges i en avslutande bibliografi. För mer specifika referenser rekommenderas att ta kontakt med något av de forskningslaboratorier eller institut som refereras ovan.

Nedan följer en uppräkning av de ur svenskt perspektiv mest relevanta konferenserna och tidskrifterna.

- IJCAI (International Joint Conference on Artificial Intelligence), ges varje udda år, varannan gång i USA, varannan gång i Europa.
- AAAI (American Association for Artificial Intelligence), ges varje år, alltid i USA.
- ECAI (European Conference on Artificial Intelligence) ges varje jämnt år, närmast 1988 i Tyskland.
- Workshop om expertsystem och dessas tillämpningar, ges årligen i Avignon.

Många europeiska länder har sina egna sammanslutningar. I Sverige finns SAIS (Svenska AI-Samfundet), som har en årlig workshop, närmast våren 1988 i Umeå. Tidigare har SAIS snarast varit en sammanslutning för inom området aktiva forskare, men allt flera intressenter från industrin har blivit medlemmar under de senaste åren.

När det gäller tidskrifter är det främst de följande som kan vara av allmänt intresse:

- Journal of Artificial Intelligence
- Cognitive Science
- The SIGART newsletter, för medlemmar av denna ACMs intressegrupp
- The AAAI magazine, för medlemmar i AAAI
- The AISB quaterly, newsletter för det brittiska AI samfundet
- SAIS newsletter, för medlemmar i SAIS
- New Generation Computing
- Expert systems strategies

AVSLUTNING

Syftet med detta nummer av SISU analys har varit att ge en introduktion till vad som avses med begreppet kunskapssystem och relatera detta såväl till forskningsområdet Artificiell Intelligens som tillämpningsområdet Expertsystem. Det har varit en ambition att presentera en så nyanserad bild som möjligt av de möjligheter respektive begränsningar som kunskapssystem i dag innebär. Det är författarnas övertygelse att denna teknologi på ett avgörande sätt kommer att påverka informationsteknologi utvecklingen, men att mycket kan vinnas om all den kunskap som konventionell teknologi innehåller kan tillvaratas. Risken är annars att vinster som den nya teknologin innebär fördunklas av brister i fråga om funktioner som blivit en självklarhet vid utvecklingen av konventionella system.

Det bästa avslutande råd som kan ges är att i varje läge använda den enklaste teknologi med vars hjälp den aktuella problemställningen kan lösas. Detta kan synas självklart, men informationsteknologi-området och inte minst användningen av kunskapssystemteknologi, uppvisar många flagranta exempel på motsatsen.

Ni som har frågor vad gäller det som presenterats i detta nummer är välkomna att vända er till någon av författarna, Carl-Gustaf Jansson eller Åsa Rudström, som båda är verksamma vid institutionen för Data och Systemvetenskap vid KTH/Stockholms universitet.

APPENDIX: KOMMENTERAD BIBLIOGRAFI

Detta appendix avser inte att helt täcka all typ av litteratur inom området, utan snarare att ge några tips om vad som kan vara lämpligt att läsa för att närmare sätta sig in i de idéer som behandlats i artikeln. För att sätta sig in i bakgrunden till den typ av frågeställningar som behandlats kan man t.ex gripa sig an med ett närmast klassiskt verk, som sedan något år dessutom finns på svenska:

Gödel, Escher, Bach - An Eternal Golden Braid, av Douglas Hofstadter, Vintage Books, NY, 1980.

Dessutom refererades vi i ett tidigare avsnitt till Polyas bok:

How to Solve It, av G. Polya, Princeton University Press, 1973

För en mer praktiskt översikt av AI-området finns en handbok i tre delar, som behandlar i princip allt som gjorts under rubriken AI fram till 1981:

Handbook of Artificial intelligence, Vol 1 - 3, av Edmund Barr och Edward Feigenbaum, William Kaufman, Los Altos, California, 1981.

Handbook är närmast att betrakta som ett uppslagsverk, men det finns också ett stort antal mer lättillgängliga introducerande böcker om Artificiell Intelligens. Vi rekommenderar varmt Rich's bok, trots att det kanske finns liknande böcker som är mer kända.

Artificial Intelligence, av Elaine Rich, McGraw-Hill, Singapore, 1983.

För en kritisk analys av AI som forskningsområde finns t.ex följande två böcker:

Artificial Intelligence and Natural Man, av Margaret Boden, Basic Books, New York, 1977.

Computer Power and Human Reason, av J. Weizenbaum, W.H. Freeman, San Fransisco, 1976.

När det gäller mer avgränsade områden, som kunskapsrepresentation, problemlösning, expertsystem, programmering och maskininlärning kan vi rekommendera följande böcker:

Kunskapsrepresentation och problemlösning

Readings in Artificial Intelligence, av Brachman och Levesque, Morgan Kaufman Publishers Inc., Los Altos, CA, 1986.

Conceptual Structures - Information Processing in Man and Machine, av John Sowa, Addison-Wesley, 1984.

Logical Foundations of Artificial Intelligence, av M. Genesereth och N. Nilsson, Morgan Kaufman, 1987.

Brachman/Levesque är en sammanställning av ett antal artiklar inom ämnet. Sowa tar upp olika aspekter av kunskapsrepresentation och för fram en representationsform som är en variant av associativa nätverk. Nils Nilsons bok är den senaste i raden av hans klassiska försök att beskriva den teoretiska kärnan till ämnet AI.

Expertsystem och hjälpmedel

A Guide to Expert Systems, av Donald Waterman, Addison-Wesley, 1986.

Introduction to Expert Systems, av Peter Jackson, Addison-Wesley, 1986.

Rule Based Expert Systems: The MYCIN Experiments of the Heuristic Progamming Project, av B Buchanan och E Shortliffe, Addison-Wesley, 1984.

Waterman är till största delen en uppräkning av vilka expertsystem som finns idag. Jackson har mer formen av en lärobok, där olika aspekter av expertsystem tas upp, ofta i form av beskrivningar av specifika system. Båda dess böcker talar dessutom en hel del om olika verktyg. Buchanan/Shortliffe är en bastant bok på över 700 sidor, och beskriver såväl själva systemet MYCIN som de teorier som legat till grund för utvecklingen av systemet. Jacksons bok inleds dessutom med några grundläggande avsnitt om kunskapsrepresentation och problemlösning, vilket även gör den till en lämplig introduktion till AI.

Programmeringsmetodik

Structure and Interpretation of Computer Programs, av H Abelson och G Sussman, MIT press, Cambridge MA, 1984.

The Art of Prolog, av Leon Sterling och Ehud Shapiro, MIT press, Cambridge MA, 1986.

Abelson/Sussman beskriver programmeringsspråket SCHEME och funktionell programmering i detta språk, men även objektorienterad programmering och logikprogrammering introduceras i termer av SCHEME.

Sterling/Shapiro uttrycker på ett klart sätt distinktionen mellan logikprogrammering och programmering i Prolog. Dessutom ges ett stort antal exmpel av lite större komplexitet.

Maskininlärning

Machine Learning - An Artificial Intelligence Perspective, av T. Michalski, J. Carbonell och T. Mitchell, Tiaoga Publishing Company, Palo Alto CA, 1983.

Machine Learning - An Artificial Intelligence Perspective, volume II, av T. Michalski, J. Carbonell och T. Mitchell, Morgan Kaufman Publishing Inc., Los Altos, CA, 1986.

Dessa två böcker består av ett urval av artiklar inom området, vilka ger en samlad bild av vad som hittills gjorts inom maskininlärning.