Expertsystem

Kandidatavhandling / ÅA

Tuomo Lyyrtö, 2010

Expertsystem

1. Vad är expertsystem?

Expertsystem är ett delområde inom artificiell intelligens. Ett expertsystem är ett datorprogram som efterliknar en experts problemlösningsförmåga och -metoder genom att fångar expertkunskap inom ett specifikt område. Området är ofta mycket smalt och noga definierat, men kunskapen däremot mycket djup och täckande. Ett expertsystem används då man behöver utnyttja kunskap inom området men inte har tillgång till en expert. En mycket central egenskap hos expertsystem är förmågan att förklara för användaren vilka resonemang ligger bakom ett visst svar. Detta är något användaren förväntar sig för att kunna lita på slutsatsen systemet dragit. Denna förmåga att förklara resonemanget bakom påståenden och slutsatser är densamma som man skulle förvänta sig av en expert. Ett expertsystem försöker alltså imitera en experts beslutfattningsförmåga. Winston() diskuterar om vilka problem som lämpar sig för lösning med expertsystem, med tanke på att de medför en avsevärd investering av både pengar och arbetstid. Han påpekar att försök att lösa problem som inte är lämpliga för denna teknologi kan leda till pinsamma och dyra misslyckanden. Vidare räknas upp ett antal kriterier som forskare har lagt upp för att hjälpa avgöra huruvida ett problem möjligen kunde lösas med ett expertsystem. Man bör också väga ifall behovet för ett expertsystem berättigar den stora investeringen. Inom områden såsom gruvindustri, affärsliv, försvar och medicin anses det finnas ett stort potential för besparing av pengar, tid och människoliv. Det nämns också att ifall ett problem går att lösa med en mera traditionell datortillämpning så är denna att föredra. Vidare sägs att problemområdet bör vara välkännt och att lösningarna inte skall kräva mänskligt förnuft. Speciellt tekniska problemområden anses vara ypperliga kandidater för expertsystemlösningar, ty de är i allmänhet välstuderade och definierade. Däremot är problem som kräver mänskligt förnuft och kreativitet svåra att lösa maskinellt. Till sist nämns att problemområdet inte får vara för utsträckt och att det måste finnas experter inom området som vill och kan dela med sig av sin kunskap för utveckling av ett expertsystem. (Winston (), Clark())

2. Bakgrund och historia

2.1. AI och expertsystem genom tiderna

Alan Turing (1912-1954) var en av de mest inflytelserika forskarna inom artificiell intelligens under vetenskapsområdets tidiga år. År 1937 presenterade Turing koncepter för Turing Universal Machine i en studie om beräkneliga tal. Turingmaskinen är en teoretisk automat som kan utföra alla möjliga algoritmer och efterlikna vilken som helst dator. Turing uppfattade att det finns vissa typer av uträkningar som maskiner inte kan lösa. Trots denna begränsning var han övertygad om att det skulle gå att utveckla maskiner som kan tänka. År 1950 föreslog han ett test för att känna igen artificiell intelligens. I turingtestet diskuterar en domare skriftligt med både en människa och en maskin. Om domaren inte lyckas skilja mellan dem, har maskinen klarat testet, och har då enligt Turing visat intelligens. År 1945 skrev John von Neumann (1903-1957) en studie där han presenterade den grundarkitektur för datorer som används än idag. I von Neumanns arkitektur lagras data och programinstruktioner i minnet och instruktionerna exekveras seriellt. Det första AI-programmet anses vara Logic Theorist, som skrevs av Allen Newell, Herbert Simon och J.C. Shaw vid Carnegie Institute of Technology under åren 1955-1956. Det utvecklades för att påvisa teorem, bl.a. genom användning av regler. Programmeringsspråket IPL (Information Processing Language) utvecklades för detta ändamål och använde sig av pekare. Begreppet artificiell intelligens sägs ha använts för första gången år 1956 av John McCarthy. Han föreslog då ett två månader långt möte som kom att äga rum vid Dartmouth University. Bland de inbjudna var många av tidens främsta forskare inom området. General Problem Solver, den allmänna problemlösaren, utvecklades år 1957 av Herbert Simon, J.C. Shaw och Allen Newell. Som en praktisk lösning anses GPS inte ha varit speciellt nyttig, p.g.a. att problemställningen och stora mängrer data måste matas in i ett mycket specifikt format. Denna problemdefiniering var arbetsdryg och slutligen var programmets insats till lösningen förhållandevis liten. År 1958 grundade John McCarthy och Marvin Minsky laboratoriet för artificiell intelligens vid Massachusetts Institute of Technology. Samma år utvecklade McCarthy programmeringsspråket LISP (List Processing) som fort kom att ersätta det tidigare allmänna AI-

programmeringsspråket IPL. Herbert Simon sägs år 1965 ha förutspåt att maskiner år 1985 kommer att klara av alla arbeten som människor klarar. Detta har senare visat sig ha varit för optimistiskt. DENDRAL, det första moderna expertsystemet utvecklades mellan åren 1965 och 1975 av Edward Feigenbaum et al vid Stanford University. Dess expertisområde var att kartlägga komplexa organiska kemikalier från data utvunnen med hjälp av masspektrometrar. Logikprogrammeringsspråket Prolog (Programmation en logique, logikprogrammering) utvecklades under åren 1971-1972 av Alain Colmerauer och Phillipe Roussel i Marseilles, Frankrike. Senare utvecklades olika varianter av språket. Prolog blev det dominanta AI-programmeringsspråket utanför USA, där LISP fortfarande höll platsen. År 1984 byggde General Electric ett expertsystem för felsökning av ellok. Systemet DELTA (Diesel Electric Locomotive Troubleshooting Aid) innehöll kunskap av experten David Smith som snart skulle gå i pension. I omkring 80% av felfallen hittade DELTA felen och kunde ge riktiga reparationsinstruktioner. Kunskapsbaser i expertsystem hade vuxit till tusentals regler och de invecklade helheterna blev svåra att hantera. Mot slutet av 1980-talet användes expertsystem allt mer inom industrin. Även andra AI-teknologier implementerades ofta tillsammans med traditionella datorprogram. Den artificiella intelligensen var därför inte så synlig, men den hade en fördelaktig inverkan på tillämpningarna. (Buchanan, Bruce G. (2010), Stottler Henke Associates (2002))

2.2. Exempel på kända expertsystem

2.2.1 Mycin

Expertsystemet Mycin utvecklades för att känna igen blodinfektioner och ge råd för vård av sådana. Mycin skrevs i LISP (List Processing) -språket. Kunskapen i Mycin består av ungefär 500 "ifall.. så.." -regler. En typisk regel i MYCIN-systemet är

```
IFALL: 1) infektionen som kräver vård är
hjärnhinneinflammation, och
2) organismens [STAIN?] är känd, och
3) organismens morfologi är känd, och
```

4) patienten har allvarliga brännskador
SÅ: Det finns svag antydning (0,3) att pseudomonas-aeruginosa är en av organismerna (förutom de som syns i odlingar och [SMEARS?]) som förorsakar infektionen.
(Clark)

Talet (0,3) i samband med slutsatsen är ett så kallat förtroendevärde (confidence/certainty factor). Talet på skalan [-1,1] är ett mått som beskriver hur starkt förhållandet mellan villkoren och dragna slutsatsen är. Varje Mycin-regel har en sådan storhet och dessa används för att räkna ut ett numeriskt förtroendevärde för den dragna slutsatsen. För att kunna behandla enskilda patienter, ställer systemet frågor gällande patientens tillstånd. Svarena behöver inte vara strikt ja eller nej, utan användaren kan också ge ett tal mellan -5 och 5. Mycin har förmågan att kombinera alla dessa osäkra fakta för att dra en slutsats vars förtroendevärde också anges. Mycins tilltänkta användarkrets var läkare. Före systemet skulle tas i bruk ens experimentellt, ville man försäkra sig om att det skulle prestera nöjaktligt. Läkare är typiskt mycket upptagna. Man var rädd att om man presenterade ett system som fungerade långsamt och gav osäkra svar, kunde det skapa fördomar som man inte senare blev av med oberoende av hur bra systemet till sist blev. Man ställde som krav att Mycin skulle komma fram till samma slutsats för vårdrekommendation som en expert, eller en alternativ, lika bra slutsats. Mycin skulle kunna avgöra ifall patienten överhuvudtaget behöver vård. Viktiga synpunkter för ett expertsystem som detta var ifall det ställer irrelevanta frågor eller lämnar viktiga frågor oställda och om det når rätt slutsats gällande bakterien som förorsakar patientens besvär. Dessa egenskaper testades genom att testa Mycin mot ett antal experter. Man gjorde en undersökning där man jämförde dess utförande med tio experter inom området. Både Mycin och var och en av experterna skulle diagnostisera 15 fall i vilka patienterna hade blodinfektioner. Resultaten visade att Mycin till största delen gav goda råd och dess kunskap närmade sig experter som specialiserat sig på området. Mycin togs dock aldrig i kliniskt bruk, men det ledde vägen för många påföljande expertsystem. (Shortliffe, Clark)

3. Kunskap - basen för expertsystem

Expertsystem är till för att lösa praktiska problem i verkliga världen, inte endast bevisa teorier. För att lösa praktiska problem behövs kunskap inom det ifrågavarande problemområdet; det räcker inte endast med att kunna härleda formler och påvisa teorem. Expertsystem är kunskapsbaserade problemlösare vars kunskap kommer från experter inom kunskapsområdet. Liksom fallet ofta är med experter, fokuserar expertsystem också på ett smalt kunskapsområde. Däremot strävar man till att kunskapen inom det valda området är möjligast komplett och djupgående. Ett expertsystem sägs vara endast så bra som dess kunskap. Expertkunskap är en kombination av teoretisk förståelse för ett problemområde och en mängd problemlösningsregler, ofta undermedvetna, som expertens erfarenhet har visat vara effektiva. Vid utveckling av expertsystem måste denna kunskap utvinnas från experter och programmeras i ett format som en dator kan tillämpa för problemlösning. Kunskapsbaserade problemlösningsmodellen som expertsystem använder sig av medför också en del problem. Kunskap föråldras, och beroende på kunskapsområdet kan föråldrandet ske plötsligt och vara radikalt. En aktiv expert kan följa med utvecklingen inom dennes kunskapsområde men en maskin måste programmeras om. Ett annat problematiskt beteende är s.k. skörhet. Det förorsakas av att expertsystemet inte egentligen är medvetet om vilken kunskap det har och vilken som det saknar. I praktiken betyder det att expertsystemet inte kan avgöra ifall det har benägenhet att besvara en ställd fråga. Det kan leda till att systemet drar en egen slutsats till ett problem vars egentliga lösning ligger utanför räckvidden av expertsystemets kunskapsbas. (Winston, Clark, Sebesta, Stottler Henke Associates)

3.1 - Kunskapen i expertsystem

Den del av ett expertsystem som innehåller expertkunskapen kallas kunskapsbas. Kunskapsbasen består av allmängiltig kunskap inom systemets expertisområde. Ofta är kunskapen framställd i form av påståenden (explicit eller deklarativ) och regler (implicit eller procedural). Explicit kunskap beskriver statiska fakta. Implicit kunskap framställs som regler enligt vilka det går att härleda kunskap från explicita fakta. Till exempel påståendet *"Tom är en man*." är ett absolut,

explicit påstående som ovillkorligen antas vara sant. Påståendena "*John är en far. Fäder är manliga.*" innehåller implicit kunskap. Att John är en far är ett explicit påstående, medan konstateringen att fäder är manliga är en allmängiltig regel som kan tillämpas på vilken som helst far. Tillsammans implicerar de att John är en man, fastän detta inte explicit påstås. Att John är en man är alltså en slutsats som kan dras från kunskapen man har om John och allmänt om begreppet faderskap.

3.2 - Kunskapsutvinning

Kunskapsutvinning (knowledge aquisition) är den arbetsprocess där man samlar kunskap om ett specifikt område för att sedan införa den i ett expertsystem. Den huvudsakliga källan för kunskap anses vara experter, personer som har lång erfarenhet av att besvara frågor och lösa problem inom ett begränsat kunskapsområde. Utvinningsprocessen är tidskrävande, och utgör en avsevärd del av hela arbetsmängden vid utveckling av expertsystem. Expertintervjuer är den mest uppenbara kunskapsutvinningsmetoden. Därutöver låter man experter lösa exempelproblem och observerar noga deras framförande. Förutom svaren och lösningarna, är man speciellt intresserad av resonemangen bakom de slutsatser experterna drar. Det är framförallt resonemangen som översätta till regler ger expertsystem deras problemlösningsförmåga. Vid planering av expertsystem är det viktigt att försäkra sig om tillgång till experter som både vill och kan dela med sig sin kunskap. Experter är i allmänhet upptagna och deras tid är värdefull. Kunskapsutvinning är krävande p.g.a. människans problemlösningsmetoder och sätt att resonera. Av den enorma kunskapsmängden som experter besitter är en stor del s.k. tyst kunskap. Denna handlar om handlingsmönster som experten tillägnat sig genom erfarenhet och inte nödvändigtvis medvetet känner igen. Sådan kunskap är svår att komma åt spontant och sedan att klä i ord för att vidare modelleras som regler. I många fall har en person svårt att exakt förklara vilken kunskap denne använt för att lösa ett problem. En människa är inte heller medveten om exakt vilken kunskap hon har och vilken inte. Ingen expert har heller all kunskap inom ett område. Därför är det önskvärt att utvinna kunskap från flera experter. Denna egenskap att sammanslå många experters kunskap är en av expertsystemens styrkor. Böcker och rapporter kan också erbjuda värdefull

kunskap för expertsystem. (Clark & McCabe (1984), Milton (2003), Winston(1983))

3.3 - Kunskapsprogrammering, kunskapsframställning

Kunskapsprogrammering (knowledge engineering) är arbetet att framställa den utvunna kunskapen i en form som datorn kan utnyttja för problemlösning. Kunskapsprogrammeraren (knowledge engineer) känner till programmeringsspråket och vet hur kunskapen framställs effektivt. Däremot är denne i ofta okunnig inom kunskapsområdet som behandlas. Kunskapsprogrammeraren arbetar tillsammans med experten och hjälper denne att utrycka sig på ett sätt som tillåter kunskapen att fångas i en kunskapsbas. Det anses tillochmed vara fördelaktigt att kunskapsprogrammeraren inte känner till kunskapsområdet. Då kan denne med större sannolikhet upptäcka bristfälligheter i kunskapen som kan förorsaka avbrått i resoneringskedjan. I ett expertsystem framställs expertkunskap i form av fakta (explicit), och regler (implicit). (Clark & McCabe (1984), Milton (2003), Winston(1983))

4. Logikprogrammering

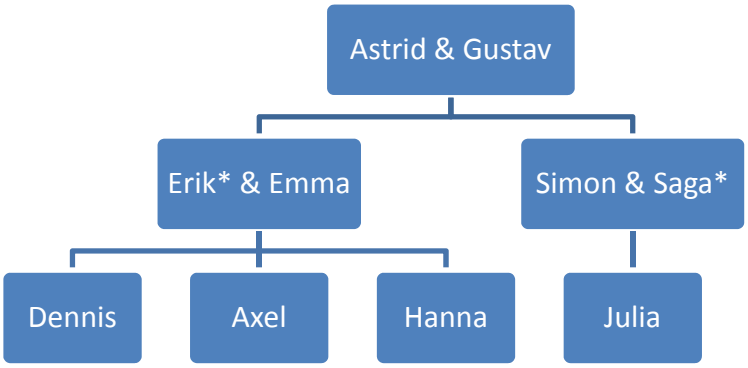
Logikprogrammering är ett programmeringsparadigm som baserar sig på matematisk logik. I detta paradigm byggs logikprogrammet upp genom att specifiera förhållanden mellan datavärden. Under exekvering besvarar sedan programmet på ställda frågor. Användargränssnittet är ofta en interaktiv tolk. Genom att besvara frågorna, kan programmet bekräfta vissa förhållanden, eller förkasta dem. Ett logikprogram kan alltså utvinna implicit kunskap från en kunskapsbas med explicita fakta och regler. (Lu & Mead)

4.1 Programmeringsspråket Prolog

4.1.1 Deklarativ kunskap

Fakta beskrivs i Prolog med predikat. Som exempel kan man tänka sig ett litet expertsystem som har som kunskap ett familjeträd som föreställer medlemmar i en släkt och deras släktskapsförhållanden sinsemellan. Konstanter betecknas i Prolog med små begynnelsebokstäver. I detta fall är personernas namn konstanter.

För att lagra den väsentliga informationen om kön, kan man lämpligen skriva ut personerna som predikat, där predikatets namn beskriver könet och argumentet personens namn. Exempelsläktträdet syns nedanom.



(släktträd för Prolog-exempel. Asterisk (*)betecknar en ingift person)

Personernas namn och kön är faktum vilka i Prolog betecknas med predikat:

```
kvinna(astrid)
kvinna(emma)
kvinna(saga)
kvinna(hanna)
kvinna(julia)
man(gustav)
man(erik)
man(simon)
man(dennis)
man(axel)
```

Dessa predikat beskriver att personerna tillhör någondera relation kvinna eller man. I detta fall kan systemet besvara endast frågan ifall en person är av ett visst kön, t.ex. kvinna(julia) resulterar i TRUE. Likaså, frågan man(X) resulterar i en lista på män i släkten. Dessa fakta finns explicit skrivna i kunskapsbasen, och systemet behöver inte använda någon slutledningsförmåga för att bevisa dessa fakta. Vissa grundsläktskaper måste också anges explicit, såsom förälderskap. Detta görs genom 2-tupelpredikat. Här kallas relationen foralder; det första argumentet är föräldern och det andra är barnet.

```
foralder(astrid,emma)
foralder(gustav,emma)
foralder(astrid,simon)
foralder(gustav,simon)
foralder(emma,dennis)
foralder(emma,axel)
foralder(emma,hanna)
foralder(erik,dennis)
foralder(erik,axel)
foralder(erik,hanna)
foralder(saga,julia)
```

Prolog har ingen uppfattning om betydelsen hos konstanterna och predikatnamnen som används, och därför har programmeraren friheten att bestämma deras betydelse. Predikaten ovan kunde lika väl betyda t.ex. att emma är astrids förälder, eller att de har samma förälder. (Seb) Det väsentliga är logiken hålls konsekvent. N-tupelpredikat satisfieras då målet överensstämmer med alla argument i ett predikat i kunskapsbasen. Exempelvis är foralder(erik,hanna) således true. Dessa predikat beskriver förälderskaperna i exempelsläkten. Fakta framställd som n-tupelpredikat fastställer att en viss sammansättning värden ovillkorligen satisfierar predikatet. [cpl]: Framställning av deklarativ kunskap, eller explicit fakta är inte det huvudsakliga ändamålet i ett expertsystem. Det speciella med expertsystem är att en avsevärd del av kunskapen förekommer som procedural kunskap, i form av regler. (Sebesta, Matthews, Lu & Mead)

4.1.2 Procedural kunskap

Procedural kunskap är kunskap som inte är explicit utskriven i programmet men kan härledas från den explicita kunskapen genom att följa regler. Regler är allmängiltiga och kan tillämpas på alla predikat som uppfyller regelns villkor. Med en ny regel införs ofta en större mängd kunskap i programmet än med ett enskilt explicit faktum. Från exemplet med släktträdet kan man genom logisk slutledning och allmän kännedom om släktskaper komma åt en stor mängd kunskap utöver den som finns explicit uttryckt med predikat. I regler används variabler som argument. Detta är nödvändigt för att kunna tillämpa regler på olika

fakta. I Prolog betecknas variabler med storbegynnelsebokstav. Följande exempel illustrerar ett par regler som definierar släktskaperna moderskap och faderskap.

```
mor(X,Y) :- foralder(X,Y), kvinna(X)
far(X,Y) :- foralder(X,Y), man(X)
```

Reglerna kan utläsas som X är Y:s moder (respektive fader) om X är y:s förälder och X är kvinna (respektive man). Med dessa regler besvarar expertsystemet TRUE på frågan mor(saga,julia) eller far(gustav,emma). Regler är verkligen styrkan bakom expertsystem. Med några regler går det att framställa stora mängder implicit kunskap från faktum. Storförälderskap, mormoderskap, morfaderskap, farmoderskap och farfaderskap kan också uttryckas med var sin regel.

`storforalder(X,Z) :- foralder(X,Y), foralder(Y,Z)` utläses som X är Z:s storförälder om X är Y:s förälder och Y äs Z:s förälder.
`mormor(X,Z) :- mor(X,Y), mor(Y,Z)` utläses X är Z:s mormor om X är Y:s mor och Y är Z:s mor.
`farmor(X,Z) :- mor(X,Y), far(Y,Z)` utläses X är Z:s farmor om X är Y:s mor och Y är Z:s far.
`morfar(X,Z) :- far(X,Y), mor(Y,Z)` utläses X är Z:s morfar om X är Y:s far och Y är Z:s mor.
`farfar(X,Z) :- far(X,Y), far(Y,Z)` utläses X är Z:s farfar om X är Y:s far och Y är Z:s far.

Denna typs kunskapsframställning är kompaktare och överskådligare än om all denna kunskap skulle uttryckas separat för varje storförälderskapsförhållande i släkten. Vidare kan man framställa förhållandet syskonskap, systerskap och broderskap.
`syskon(X,Y) :- foralder(Z,X), foralder(Z,Y), X \=`
`Y` utläses `X` är `Y`:s syskon om `Z` är `X`:s förälder och `Z` är `Y`:s förälder, och `X` är inte `Y`.
`syster(X,Y) :- kvinna(X), syskon(X,Y)` utläses X är Y:s syster om

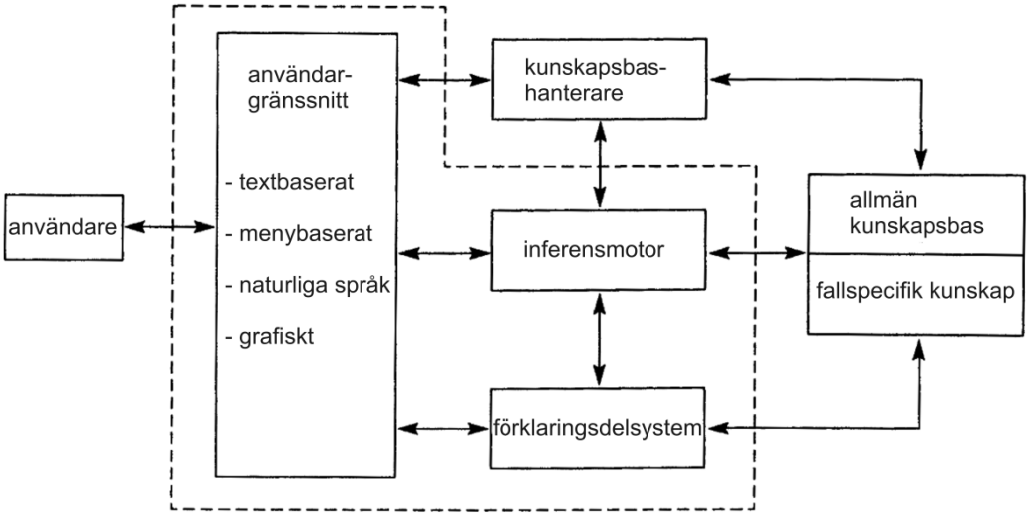
X är kvinna och X är Y:s syskon.
`bror(X,Y) :- man(X), syskon(X,Y)` utläses X är Y:s bror om X är man och X är Y:s syskon.
(Sebesta, Matthews, Lu & Mead)

4.1.3 Slutsatsdragning (inferencing)

Prolog är ett deklarativt programmeringsspråk. Språk som används för att i detalj beskriva algoritmer, d.v.s. där programmeraren skriver ut hur ett visst resultat nås kallas procedurala. Förfrågningar (queries) kan ställas Prolog via dess textbaserade användargränssnitt. I Prolog ställer man frågor, eller sk. mål (goals) som programmet försöker bevisa. Om det inte går, förkastas de. Till exempel kan man ställa frågan man(gustav). Uttrycket är identiskt till predikatet som fastställer detta faktum i programkoden, och Prolog ger då som svar TRUE. Ett jakande svar betyder att kan bevisas enligt den information som är införd i kunskapsbasen. Däremot betyder ett nekande svar att påståendet inte kan bekräftas, inte att det är absolut falskt. Om man ställer frågan `mor(hanna,kasper)`, svarar Prolog med FALSE. Detta betyder inte nödvändigtvis att det går att bevisa vara osannt, utan att påståendet inte kan bevisas sant t.ex. om den efterfrågade informationen inte finns. Prolog utför slutlsatsdragning (inferencing) för att nå de mål man ställer. Genom slutsatsdeagning kommer Prolog åt implicit kunskap som är framställd med hjälp av regler. För att bevisa ett mål, måste Prolog hitta en kedja av regler och fakta mellan det ställda målet och kunskap i kunskapsbasen. Vid slutsatsdragning instantieras variablerna i regler med värden och uttrycken jämförs med kunskapsbasens innehåll. Om man vill veta ifall emma är en mor, kan man skriva `mor(emma,_)`. Understrecket som argument satisfierar alltid regeln för det argumentets del. Prolog beaktar regeln `mor(X,Y) :- foralder(X,Y), kvinna(X)` och börjar med att instantiera X med värdet emma. Därefter försöker Prolog satisfiera `foralder(emma,_)` och `kvinna(emma)`. Dessa predikat satisfieras, ty de finns explicit utskrivna i kunskapsbasen. (Sebesta)

5. Implementering av expertsystem

5.1 Expertsystemskal



figur som förklarar expertsystemets uppbyggnad (Winston)

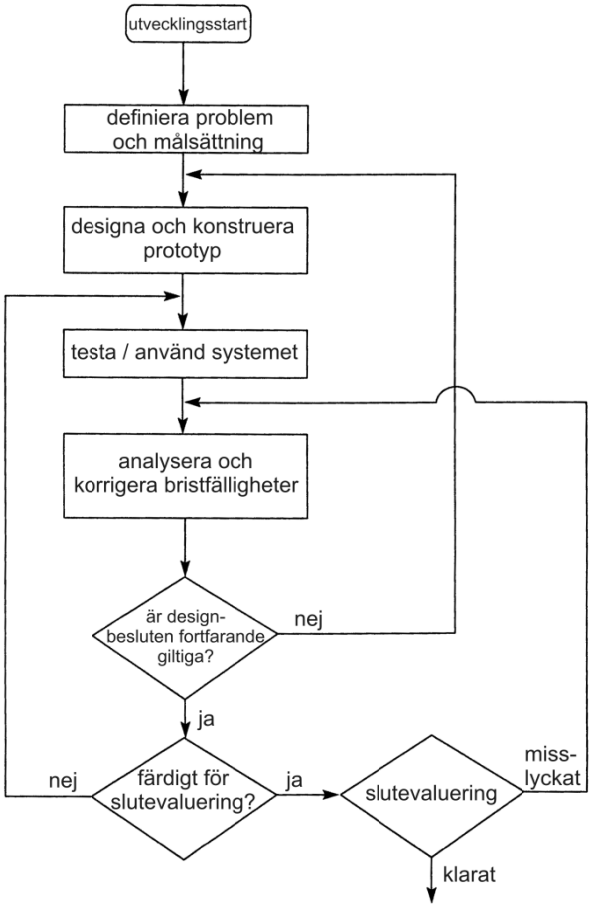
Expertsystemskal (expert system shell) är som namnet antyder ett "tomt" expertsystem. Delarna omringade av streckade linjen i figuren ingår i expertsystemskalet. Separering mellan kunskapsbasen och programmets andra delar möjliggör att expertsystemskal kan användas för att spara utvecklingsarbete och förkorta utvecklingstiden. Endast kunskapsbasen måste byggas upp för att täcka det tilltänkta kunskapsområdet. Därmed kan inferensmotor, förklaringssystem och användargränssnitt återanvändas som grund för ett nytt expertsystem. Exempel på expertsystemskal är CLIPS utvecklat av NASA och JESS (Java Expert System Shell) utvecklat av Sandia National Laboratories. Winston, Friedman-Hill, Ernest J. (2001)

Kunskapsbashanteraren hjälper programmeraren att korrigera påträffade fel som ofta kan lokaliseras med hjälp av information från förklaringsdelsystemet. Kunskapsbashanteraren kan också hjälpa vid införning av ny information genom att övervaka syntaxet och köra tester som för att undvika konflikter med tidigare kunskap.

Ett expertsystem kan ett delsystem för kunskapsbashantering. Kunskapsbashanteraren finns för att hjälpa programmeraren att korrigera

påträffade program- eller kunskapsfel som kan lokaliseras med hjälp av information från förklaringsdelsystemet. Vid införning av ny kunskap i kunskapsbasen kan kunskapsbashanteraren vara till nytta genom att övervaka syntaxet och utföra tester för att undvika konflikter med tidigare kunskap. (Winston)

5.2 Utvecklingsprocessen



utvecklingsprovessen använd för expertsystem (Winston)

Enligt Winston spelar kunskapsprogrammerare, expert och kund stora roller i utvecklingen av expertsystemet. Kunden, eller slutanvändaren ställer vissa designbegränsningar och krav som systemet måste uppfylla. Experten fungerar som kunskapskälla (kunskapsutvinning behandlades i 3. kapitlet). Programmeraren känner till programmeringsspråket och ansvarar för val av mjuk- och hårdvaroverktyg för projektet. Dessutom samarbetar programmeraren med experten för att hjälpa denne att uttrycka sin kunskap på ett sätt som kan överföras i en korrekt och effektiv kunskapsbas. Utvecklingen av expertsystem sker med hjälp av en tidig prototyp vars kunskapsbas stegvis utvidgning. Prototypen som skapas klarar i början av att lösa problem inom ett mycket smalt område. Den fungerar som en indikation att de gjorda designbesluten är giltiga. Därefter kan kunskapsbasen utvidgas stegvis genom att låta systemet lösa problem och korrigera brister där de upptäcks. Kunskapsbasen utvidgas alltså stegvis, den byggs inte upp på en gång. Ifall designbesluten visar sig lyckade, kan prototypens utvidgning fortsätta stegvis tills

det utvecklas till det slutliga systemet. Ett expertsystem behöver inte heller nödvändigtvis anses färdigt efter den egentliga utvecklingsprocessen. En kunskapsbas kommer alltid att ha brister och kunskapsbasens komplettering skall anses som en pågående process under systemets hela livslängd. (Winston)

Källförteckning

Böcker

Castillo, Enrique & Alvarez, E. (1991). *Expert Systems: Uncertainty and Learning*. Springer, 1st Edition

Clark, K.L. & McCabe, F.G. (1984). *micro-PROLOG: Programming in Logic*. Logic Programming Associates, Ltd.

Patrick, Henry Winston (1984). *Artificial Intelligence*. Addison-Wesley, 2nd Edition.

Sebesta, Robert W. (1993). *Concepts of Programming Languages*. The Benjamin/Cummings Publishing Company, Inc, 2nd Edition.

Dokument

Lu, James & Mead, Jerud J. *Prolog A Tutorial Introduction*. Bucknell University, Computer Science Department.
Hämtat från
http://www.soe.ucsc.edu/classes/cmps112/Spring03/languages/prolog/PrologIntro.pdf i mars 2010.

Shortliffe, Edward H. *Mycin: A knowledge-based computer program applied to infectious diseases*. Stanford University School of Medicine, Department of Medicine.

Webresurser

Buchanan, Bruce G. (2010). *Timeline: A Brief History of Artificial Intelligence*. Hämtat
från http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/BriefHistory i mars 2010.

Friedman-Hill, Ernest J. (2001). *Jess, The Java Expert System Shell*, Version 5.2. Hämtat från http://www.jessrules.com/jess/docs/52/index.html i mars 2010.
Matthews, James (2000). *Introduction to Prolog (Part I)*. Hämtat från http://www.generation5.org/content/2000/prg00.asp i mars 2010.
Nick, Milton (2003). *Knowledge Aquisition*. Hämtat från http://www.epistemics.co.uk/notes/63-0-0.htm i mars 2010.
Stottler Henke Associates (2002). *Artificial Intelligence History*. Hämtat från http://www.stottlerhenke.com/ai_general/history.htm i mars 2010.
Uluoglu, Belkis (2006). *Declarative / Procedural Knowledge*. Hämtat från http://www.designophy.com/designpedia/design-term-1000000001-declarative-.-procedural-knowledge.htm i mars 2010.