Tekniska Högskolan i Lund
Avdelningen för Bärande Konstruktioner

Lund Institute of Technology
Department of Structural Engineering

EN TILLÄMPNING AV ARTIFICIELL INTELLIGENS MED PROGRAMSPRÅKET PROLOG.

EXAMENSARBETE
Handledare: Alberto Herrera

LUND DECEMBER 1988

Kalle Trulsson
Thomas Björklund

INNEHÅLLSFÖRTECKNING

INNLEDNING / MÅLSÄTTNING

Vi engagerades av Bärande konstruktioner LTH hösten 1986 för att med hjälp av en ny datorteknik, Artificiell Intelligens AI, skapa någon form av hjälpmedel för byggbranschen. Efter en del inledande orientering inom AI började vi inrikta oss mot det logiska programspråket Prolog i syfte att skapa ett datorprogram. Vi ville i första hand skapa ett användarvänligt program. Detta innebär enkla indatarutiner som talar om när indata är orimligt eller felaktigt. Till skillnad mot sådana program där exekveringen avbryts vid fel indata. Man bör också enkelt kunna ändra indata utan att behöva börja om programmexekvering och dessutom kunna utelämna vissa indata.

Efter att ha införskaffat en prologkompilator återstod att välja vad som skulle programmeras. Vi ville skapa ett program som kunde dimensionera de flesta olika betongkonstruktioner, typ pelare, plattor, skivor m.m.,och började med en rektangulär balk. Det visade sig att denna enda balk var en lagom begränsning eftersom programmeringsinitieringen blev så omfattande. Att börja var svårt, att fortsätta skulle vara lättare.

Denna uppsats färdigställdes hösten 1988.

Vi vill särskillt tacka följande för hjälpen.

* Alberto Herrera, vår handledare och Per Christiansson, vid avdelningen för bärande konstruktioner LTH, för deras stöd och uppmuntran under arbetets gång.
* Paula för hjälp med redigering.
* Byggkonsult Anders Högberg AB för lån av CAD vid uppritandet av bilder.

SAMMANFATTNING

Artificiell Intelligens är en vetenskap som utvecklas snabbt och kommer allt mera till nytta inom olika områden. En viktig tillämpning av AI-tekniken är s.k. KunskapsBaserade System. KunskapsBaserade System (KBS) är ett särskilt värdefullt hjälpmedel för att skapa datorprogram som klarar ostrukturerade problem. Det vill säga sådana problem som måste lösas med tumregler och antaganden och ej kan beräknas strikt numeriskt. För att framställa ett KBS bör en viss arbetsgång användas. Först bör en s.k. konceptuell modellering av problemet göras och därefter kan denna modell datorrepresenteras. Den konceptuella modelleringen är mycket viktig för att nå ett bra resultat. Med hjälp av denna skapas ordning i den dolda oredan i problemet och den implicita kunskapen friläggs. Vid datorrepresentation kan i princip alla tillgängliga programspråk användas lämpligast är dock s.k. AI-språk typ Lisp och Prolog. Dessa AI-språk utnyttjar en teknik, kallad logisk programmering, som närmar sig det mänskliga logiska tänkandet. Med logisk programmering behövs inga utförligar instruktioner ges till datorn hur ett givet problem skall lösas. Istället skall problemet beskrivas och tillgänglig kunskap friläggas. Programmet exekveras genom att ge en logisk fråga som programmet med sin etablerade kunskap sätts att bevisa.

Vårt exempel på kunskapsbaserat program utför dimensionering aven rektangulär betongbalk i brottgränstillstånd. Programmet ger stöd vid dimensionering genom att själv välja betonkvalitet, armeringskvalitet, stångdimension samt antal armeringslager. Kunskapen som programmet innehåller består både av de rent matematiska sambanden vid dimensionering samt de regler som styr vilka kvaliteter/ dimensioner/ antal lager som kan väljas. Till stöd för beslut förfogar programmet över en databas med existerande alternativ. En konceptuell modellering är utförd och visas i ett flödesschema. Detta schema är dock till viss del raserat vid datorimplementationen p.g.a. tekniska skäl.

Att skapa mer användarvänliga program med Prolog visade sig relativt enkelt. Speciellt varianten "Turbo Prolog" vilken vi använde är enkel att arbeta med när man väl har trängt instrukturen. Språket ger goda möjligheter att prova alternativa lösningar och bygga på utan att behöva göra stora ingrepp.Dessutom finns en mängd finesser som inbjuder till att skapa snabba, användarvänliga indatarutiner. Svårigheterna ligger i att utföra en riktig konceptuell modullering. När det-

ta väl är gjort kan man överlåta till Prolog att lösa viss programmering d.v.s.vissa problem behöver ej beskrivas i detalj vid implementeringen. Detta gäller t.ex. sökningsrutiner vilka är lösta en gång för alla med s.k. backtracking. Denna backtracking funktion gör det enkelt att koppla ihop ett program med en databas. Numeriska beräkningar utförs bättre med språk typ Pascal och Fotran än med Prolog. Men Prolog erbjuder goda möjligheter för interaktion med andra programspråk.

Prolog underlättar programmeringsarbetet med hjälp av sin enkla struktur och sitt minimala krav på formalia. Prolog närmar sig det mänskliga tänkandet några steg. Valet av rätt problemområde för implementering är dock av största vikt för att få etteffektivt utnyttjande av Prologspråkets fördelar. Viktigt är också att en grundlig konceptuell modellering utförs.

Några ord på vägen.

Kunskapsområde som lämpar sig för KBS bör :

* vara klart avgränsat.
* vara delbart i delproblem och kunna representeras i beslutsträd.
* inte omges av nogranna lagar och/eller matematiska regler.
* bygga på empirisk kunskap.

1 ARTIFICIELL INTELLIGENS

1.1 Vetenskapen-AI

Artificiell Intelligens (AI) som helhet är en vetenskap och på många håll i världen pågår en intensiv forskning. Denna forskningen bedrivs i huvudsak genom att "simulera" och därigenom "förstå intelligent beteende". För att åstadkomma detta har experiment med olika datormodeller bedrivits. Några av de modeller som utvecklats är:

* interaktiv databehandling.
* speciella språk. (Lisp,Prolog,etc...)
* problemlösningsmetodik. (sökningsstrategier,kunskapsrepresentation)

AI är inte en studie av datorer, utan en studie av intelligens i tanke och funktion. Eftersom dess teorier är utryckta som datorprogram används datorer som verktyg. Programmen styr maskiner att utföra invecklade operationer vilka endast intelligenta personer skulle kunna upprepa./8/

Det räcker alltså inte att konstruera en maskin som utför någonting snabbt och/eller automatiskt. Någon form av intelligens måste vara inbyggd, något som ger maskinen en medvetenhet om av vad den sysslar med och en viss förmåga att fatta självständiga beslut. AI gör för tankearbetet vad mekaniseringen och automatiseringen gör för verkstadsindustrin. Svårigheterna med användandet av AI är att det kan vara svårt att hitta de regler som styr automatiseringen d.v.s. de regler som styr det mekaniska förloppet. /3/

Denna sista del leder in på området hur och på vilket sätt AI-teknik utnyttjas. Detta kommer att belysas längre fram. (kap 2.3 och kap. 9)

Viktigt att minnas i dessa sammanhang är det faktum att ingen maskin blir smartare än dess skapare, möjligen snabbare. Efter en intensiv utveckling under cirka 20 år inom AI-baserad spelteknik, t.ex Spaceinvaders har många olika spelsystem utvecklats. I ett januarinummer 1988 i tidning en Ny Teknik kunde man läsa att en tävling mellan olika spelsystem utlysts. Spelvarianten var ett gammalt japanskt spel kallat GO, vilket något liknar vårt västerländska schack. När täv-

lingen var avgjord fick det vinnande spelet möta en tolvårig pojke. Efter en halvtimme blev datorspelet besegrat. Tänka vad man vill om detta.

1.2 Vad är intelligens ?

För att få ett grepp om vad AI-tekniken försöker simulera redovisas här några definitioner av intelligens,hämtade ur AI-litteraturen.

I en enkel definition beskrivs intelligens som förmågan att kunna utföra t.ex. ett byggnadsverk under en någorlunda kort tid. Exempelvis byggde mäniskorna i Egypten pyramider snabbare än vad koralldjuren bygger korallrev. Intelligens innebär inte bara att utföra någonting snabbt utan även att vara medveten om vad man gör och därför kunna utföra något snabbt.

Medvetenhet innebär en förmåga att kunna dra lärdom från tidigare erfarenheter. Alltså en förmåga att kunna dra egna slutsatser utifrån insamlade kunskaper, både erfarenhetsmässiga och akademiska, se fig. 1.2.

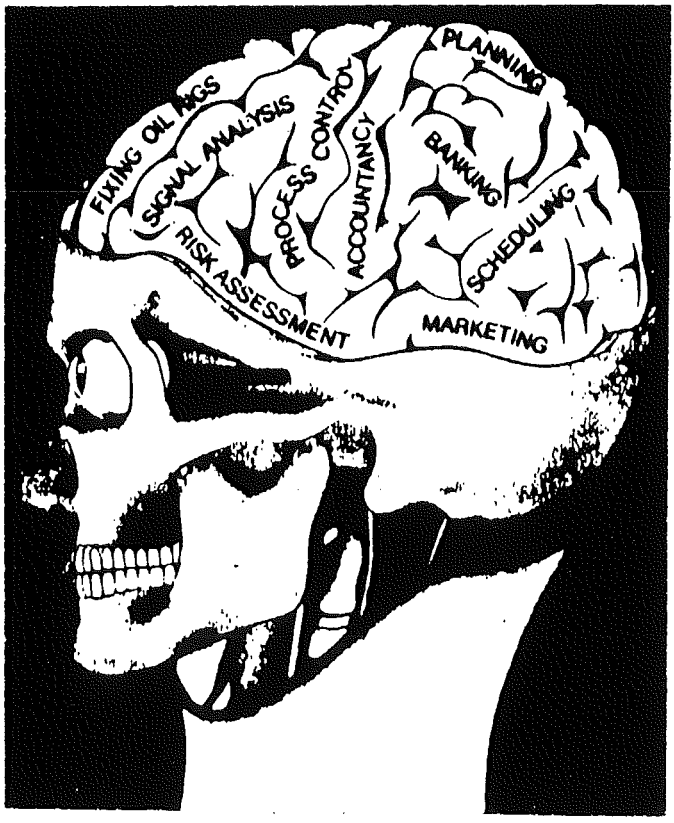


Fig. 1.2 Intelligens innebär rationell hantering av kunskap och information./4/

Intelligens är förmågan att använda, manipulera och ta tillvara kunskap och utifrån detta beskriva den beslutsprocess som genererar en lösning. /1/

Att beskriva intelligens är väldigt komplext och kan svårligen definieras konkret. Lite enklare är det att beskriva de förmågor som intelligent beteende innebär. Detta är gjort i en skrift som beskriver uppbyggnaden av expertsystem /6/.

Intelligens innebär följande egenskaper vid handhavande av information och kunskap och de skall på något sätt fångas i ett datorprogram inom AI-området:

* urval (kunna finna vad som är relevant)
* memorering (kunna komma ihåg lärdom)
* kombination (kunna förstå samband och mening)
* uppskattning (kunna finna trolig lösning)
* induktion (kunna resonera från fakta till en allmän lösning)
* anpassning (kunna fungera i olika miljöer.)
* härledning (kunna bevisa genom logisk härledning)

1.3 AI för Ingenjörer

På kort sikt kommer AI-tekniken främst att erbjuda användarvänligare datorprogram. Tekniken kommer allt mer att fungera som en länk mellan människa och maskin. Maskinen kan i detta fall vara allt från ett expertsystem i fick format till en hel industriprocess.

Av den nya tekniken kan man förvänta sig datorer och program som kan: /7/

* understödja resonemang under problemlösning. (Genom alternativa undersökningar och fakta kombination samt möjlighet för användare att resonera med datorn.)

* förstå naturligt språk.
* ges förmåga till inlärning.
* bli mer flexibel. (Klara oförutsedda situationer och kunna hoppa mellan olika problemområden.)
* innehålla ett brett spektrum av beteende. (Ökad användbarhet, mindre risk för låsning.)

1.4 Historia

Pionjärstegen inom AI-forskningen togs i princip när de första datorerna kom i bruk. Det började med utveckling av mera generellt användbara datorer och mera användarvänliga programspråk. Ny teknik utvecklades som behandlade problemlösning, kunskapsrepresentation, kunskapsinlärning och tumregler för intelligent sökning. I början på 70-talet bildades ett stort antal olika forskningsinriktningar in begripande:

* naturligt språk.
* spel utveckling.
* teorem bevisning.
* maskinellt seende. (robottillverkning)

Fig. 1.4 Historisk vetenskapsman. /7/

I Storbritannien föddes en önskan att undfly de förhärskande numeriskt-orienterade datorapplikationerna. Många ambitiösa forskningsprojekt påbörjades. Olyckligtvis underskattades svårigheterna och tidsåtgången vid denna utveckling. Detta ledde (1972) till att AI-forskningen i Storbritannien fick dåligt anseende. Intresset svalnade och många ledande vetenskapsmän flydde utomlands, främst till USA. Här inträffade inte samma bakslag. Anseendet i akademiska kretsar bestod och militären fortsatte att bidra med pengar. I detta gynnsamma klimat utvecklades de första kommersiella expertsystemen (1980).

USA:s dominans inom AI-forskningen pågick under hela 70-talet och bröts inte förrän japanernas intåg 1982. Då publicerade Japanese Institute for New Generation Computer Technology (ICOT) ett förslag för "femte generationens datorsystem". I denna publikation föreslogs en helt ny klass av datorer. Dessa skulle innehålla en kraftfullare maskinutrustning, vara byggda med en helt ny filosofi och inriktade på nya applikationsområden. Detta nya angreppssätt av datoranvändning ansågs kunna bli verklighet på 1990-talet. Utvecklingen drevs mot de nya målen och omdirigering genomfördes från strikt nummerisk datorkörning, till datorer som kan förstå meningen med inmatad information och de problem som skall lösas. Under tiden hade en grupp brittiska och europeiska forskare utvecklat logisk programmering. Detta forskningsområde visade sig vara centralt i japanernas femte generationens datorsystem. Stort intresse visades av forskare, inom datorutveckling för femte generationens datorssystem och logiskprogrammering, vilket har lett till programspråket PROLOG.

2 KUNSKAPS BASERADE SYSTEM

2.1 Allmänt

KBS eller Kunskaps Baserade System är en datorapplikation av AI-tekniken. KBS har visat sig användbar bl.a. för att skapa expert-system. Här följer några karakteristiska målsättningar med kunskapsbaserade (expert) system /5/:

* är väl lämpade för symbolhantering.
* understödjer resonemang under problemlösning. (Erbjuder användare möjlighet att ställa frågorna varför och hur , till systemet.)
* ny kunskap kan tillfogas efterhand.
* kan hantera ofullständig eller osäker kunskap. (T.e.x. sanningar som endast är säkra med viss sanolikhet.)
* kan erbjuda system som bättre förstår naturligt språk.
* kan understödja kunskapshämtning.
* kan snabbt ge beslutsstöd i kritiska situationer.
* är väl lämpad att understödja kunskapsöverföring.

Som synes härstammar KBS från AI men målen är lite förfinade.

2.2 Uppbyggnad

Systemet kan grovt indelas i tre beståndsdelar:

* Kunskapsbas (Innehåller generell fakta, tummregler mm.)
* Kontext (Samling av symboler eller fakta som speglar aktuellt problem. Här samlas även information som genererats under varje speciell datorkörning.)
* Inferensmaskin (Denna del styr programmexekveringen genom att använda kunskapsbasen för att förändra kontexten.)

Då kunskapsbasen skapas måste en kunskapsrepresentation väljas t.ex. semantiska nät, regler, ramar, satslogik, predikatberäkning mm (se kap. 3). I kunskapsbasen placeras sådan fakta, se fig 2.2, som beskriver önskat problem och de regler som styr det samma. Faktan och reglerna kan antingen vara hämtade direkt ur faktaböcker eller vara av mer erfarenhetsmässig karaktär. Med erfarenhetsmässig menas här sådan kunskap som t.ex. en expert har förvärvat efter många års problemlösande.

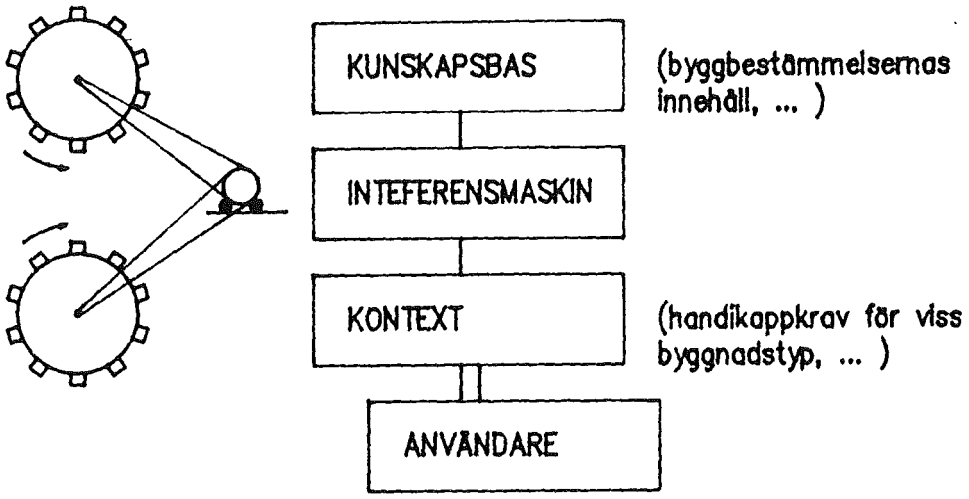


Fig. 2.2 KBS uppbyggnad.

Innehållet i kontexten består av den indata som matas in under programmexekveringens gång. Om kunskapsbasens exempelvis beskriver metodiken vid diagnoseringen av olika sjukdommar, kan kontexten bestå av de olika symptom som aktuell patient lider av. Innehållet i kontexten förändras alltså varje gång ett nytt problem skall lösas. Både fakta typ:jag har huvudvärk, och regler typ: titta endast bland tropiska sjukdommar, kan ingå i kontexten. Slutligen är det inferensmaskinen (inferens = slutledning) som styr hela exekveringen. Denna bestämmer i vilken ordning som fakta och regler i de övriga två skall kombineras. För att åstadkomma detta används olika sökstrategier. Man pratar om framåtsökning, bakåtsökning, problemreduktion o.s.v. Val av sökstrategier avpassas efter problemtyp. Därför kan ingen sägas vara bättre än en annan.

Man får lätt uppfattningen att KBS är ett statiskt system. Likt en relästation i vilken elektronerna är ersatta med fakta och regler som trillar igenom till rätt fack med sökt lösning. Det är delvis sant men transportsträckan är omöjlig att förutsäga. Ett stort KBS innehåller många regler som används vid olika tillfällen. Olika regler används vid olika körningar. Dessutom kan i vissa situationer användningen av de gamla reglerna ändras. Detta kan ge samma effekt på systemet som om ny regel hade infogats. Den sistnämnda egenskapen är väsentlig för systemets flexibilitet.

2.3 Användningsområde

Vanligt ingenjörsarbete kan huvudsakligen indelas i tre etapper.

ETAPP 1. ANALYS
Problemet som skall lösas struktureras och delas upp i hanterbara delproblem. Lämpliga modeller väljes. Modellerna skall med stöd av exempelvis matematiska samband och naturlagar väl kunna beskriva önskat problem.

ETAPP 2. BERÄKNING
Beräkning av problem utförs genom att kända fakta stoppas in i vald beräkningsmodell.

ETAPP 3. UTVÄRDERING, DIAGNOS
Resultatet av beräkningen jämförs med önskat utfall och kontrolleras så att givna krav, exempelvis normer, är uppfyllda. Om resultatet ej är tillfredställande måste ny analys utföras.

Exempel

Låt oss anta att du är en konstruktör och skall dimensionera ett hus. Uppgiften är så stor att en strukturering och uppdelning i delproblem krävs. Efter en del arbete har du gjort detta och skall nu lösa delproblemet "Vilken pelardimension ?". Du väljer modellen att räkna pelaren som ledad uppe och nere. Den första och svåraste etappen är därmed avklarad. I nästa etapp stoppar du in pelardimensionen du fått av arkitekten och utför beräkning en. Till slut, i etapp 3, kontolleras att pelarens bärförmåga klarar givna laster. Dessa laster kan i detta fall symbolisera de krav som ditt delproblem ställer. Skulle nu inte pelaren klara lasterna blir du kanske tvungen att välja en ny beräkningsmodell, exempelvis nya dimensioner, randvillkor, se fig 2.3, eller material.

Fram till nu har många datorprogram utvecklats som snabbt och säkert klarar etapp 2. Att automatisera de övriga etapperna har varit svårare. Orsaken är att i dessa krävs bedömningar och antaganden vilket en numeriskt orienterad maskin har svårt att behandla. KBS erbjuder möjligheter att hantera ofullständig kunskap exempelvis regler som baseras på antaganden och bedömningar. Därför skulle KBS kunna användas för att bygga en maskin som klara etapp 1 och 3. Denna maskin skall och kan inte helt ersätta ingenjören utan endast vara ett hjälpmedel som ger snabbare och kanske bättre lösningar. Beslut som fattas vid analysen skulle betydligt förenklas om antalet inverkande faktorer kunde

minskas. Dessa faktorer kan i byggnadsindustrin vara samhällets krav och byggherrens önskemål kombinerat med naturlagar. En expert har en förmåga att sålla bort oväsentlig fakta och därmed snabbt hitta bäst fungerande lösning. Men inte ens en expert kan hålla allt för många faktorer i huvudet samtidigt. Ett KBS skulle kunna läras att grovsålla bland faktorerna och inringa särskilda problemområden. Experten skulle därigenom i ännu större grad kunna koncentrera sig på det väsentliga. Och inte minst skulle novisen slippa mycket onödigt arbete.

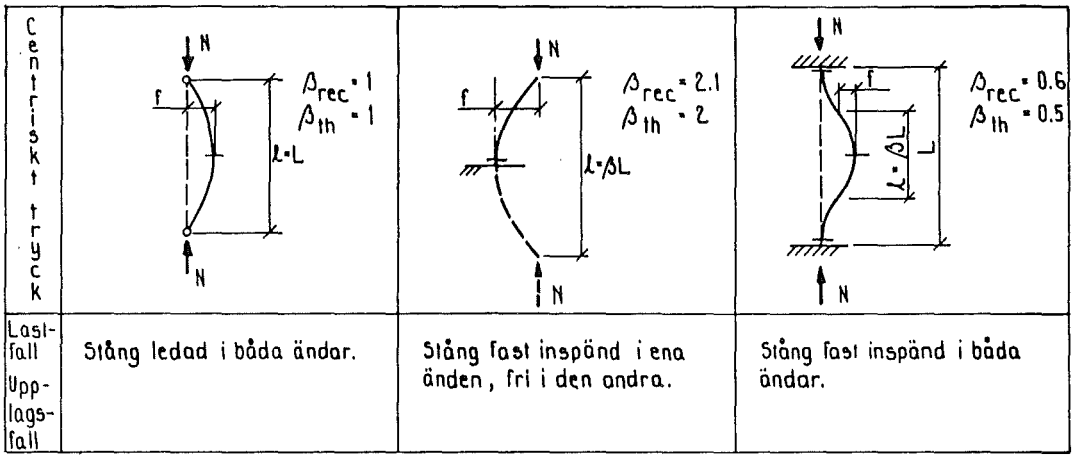


Fig. 2.3 Olika inspänningsfall vid dimensionering av pelare. /11/

När kunskaper som används vid problemlösning endast är rutin kan KBS fungera som en arbetskamrat (assistent,lärling). Om kunskapen har expertkaraktär kan KB-systemet användas för att lösa svåra problem./1/

Några viktiga områden där KBS kan fungera som hjälpmedel inom byggbranschen och andra industrier. /7/

* redovisning.
* felanalys.
* intelligent sökning i befintliga databaser.
* tolkning av normer.
* rådgivningssystem.

2.4 Allmänna kommentarer

KBS-tekniken torde vara väl lämpad för byggbranschen eftersom denna bransch, till skillnad från maskinsidan, projekterar och produerar sina produkter under relativt kort tid. Detta kräver snabba beslut och det finns relativt små möjligheter att ändra inriktning. Hela grunden kan i vissa fall vara färdig-

byggd när de sista ritningarna till taket är klara. Skulle ett stort problem i husets systemlösningen dyka upp i detta läge, är både byggare och beställare illa ute. Därför finns behov för konstruktions- och projekteringshjälpmedel som snabbt kan sondera sytemlösningar och, inte minst, kontrollera dem mot gällande normer.

Byggnadsindustrin använder heller inte exakta lösningar. Till skillnad från andra tillverkningsindustrier används överslagsmässiga och erfarenhetsbaserade beräkningar i stor utsträckning. Byggbranschens verklighet, med ett nästintill oändligt antal osäkra faktorer, klarar inte exakta fysiska modeller. Därför krävs mer arbete med problemanalys än i övrig industri.

Expertsystem som endast hittar en exakt lösning behövs inte. Det som behövs är expertsystem som hittar flertalet olika lösningar som fungerar tillsammans i en osäker omgivning. En omgivning som styrs av människans behov och begränsningar.

3 KUNSKAPSBASERAD PROGRAMVARA

Finns det ett visst tillvägagångssätt för framställning av kunskapsbaserade program ? Svaret är ja och denna analys av problemet kan naturligtvis göras på många olika sätt, men inom KBS-området är en viss arbetsgång att föredra. Det beror på att de program som framställs inom KBS-området skall eller bör ha ett beteende som är en spegelbild av mänskligt intelligent beteende. Traditionella program i t.ex. Fortran, Pascal eller C kan inte göra anspråk på att efterlikna intelligent beteende då programstrukturen är statisk och sekventiell. Programmen innehåller kunskap i form av algoritmer, vilket är en låg nivå av friläggande och analys av kunskapen. Av denna anledning ställs inte lika stora krav på strukturering av problem vid implementering i traditionell programmeringsmiljö, som vid implementeing i KBS-miljö.

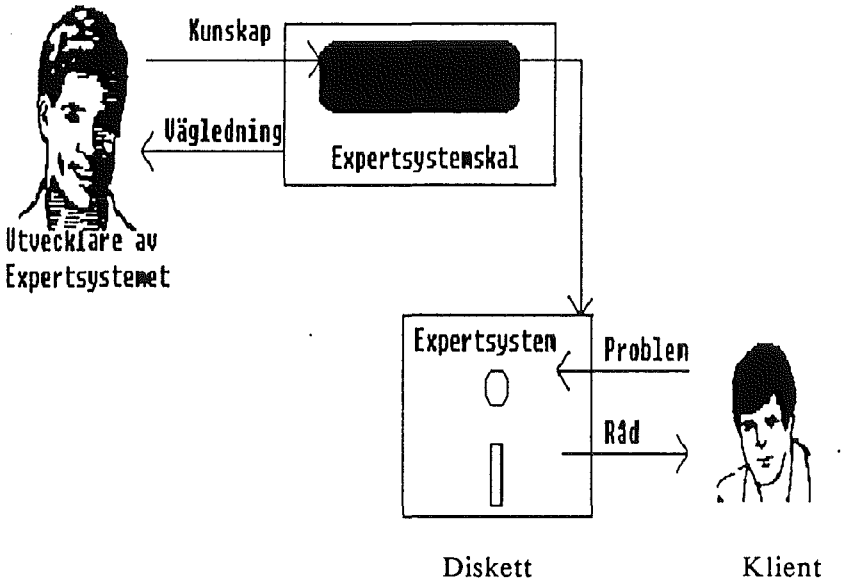


Fig. 3 Utveckling och använding av expertsystem. /8/

Tillvägagångsättet kan delas in i två faser:

1/ Konceptuell modellering av problemområde
2/ Datorrepresentation av problemområde

3.1 Konceptuell modellering

Konceptuell modellering innebär att skapa en modell av verkligheten (problemområdet) där kunskapen är frilagd och relationer etablerade mellan problemområdets minsta kunskapsbeståndsdelar.

Området kan delas in i tre underområden:

3.1.1 Formulering av problem:

Man skapar en allmänn bild av vad problemet handlar om och definierar vad man söker för resultat och vad som är givna förutsättningar.

3.1.2 Indelning av problem i delproblem:

Ett problemområde, i även sin enklaste form, är mycket komplext när man börjar undersöka dess kunskapsstruktur. Ofta kan problemet delas upp i en beslutsstruktur (trädstruktur) där huvudproblemet består av ett antal delproblem. Delproblemen måste definieras och samband etableras mellan dem såväl horisontellt som vertikalt i trädstrukturen.

3.1.3 Definition av problemlösning för delproblem:

Här kan skiljas på två problemtyper:

Analys:
Analyssituationen gäller när man är intresserad av att veta vilken respons en stimuli ger i en given fysisk omgivning ." I denna situation skapar man en idealiserad modell av det aktuella systemet./.../ Resultatet utvärderas och generaliseras till det gällande systemet om resultatet är korrekt." /3/

Design:
Designsituationen gäller när en uppsättning krav är definierade. Problemet är att bestämma ett lämpligt, ändamålsenligt system som uppfyller de ställda kraven. Man kan här ta fram flera olika systemgestaltningar för att utvärdera vilken som bäst uppfyller kraven, se fig. 3.1.3.

Skillnaden i analys och design ligger i att analysen inte befattar sig med några normer ställda från samhället. Vid t.ex. dimensionering av en stålpelare tillämpas designen där BSK's normer är de regler som styr lösandet av dimensione-

ringsproblemet. Är problemet att ta reda på pelarens snittkrafter från en given last, ja då hamnar problemet i analyssituationen. När pelaren skall dimensioneras måste även analyssituationen tillämpas då snittkrafterna är intressanta för att få dimenionerande påkänning.

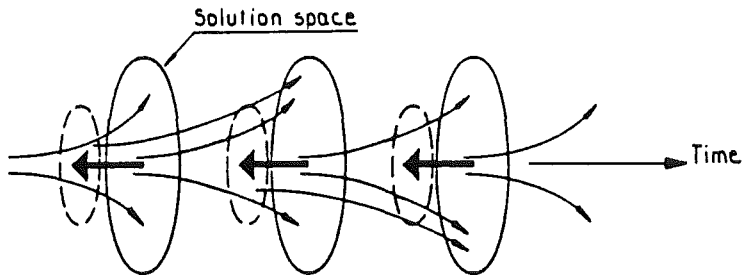


Fig. 3.1.3 Design som en utforskande process. Pilar motsatt tidsaxeln pekar mot problem mål. /9/

3.1.4 Exempel

För att illustrera den abstrakta tankegången i den konceptuella modelleringen visar vi ett enkelt exempel.

Formulering av problem: Jag är sugen på att äta sockerkaka. Jag vill baka den själv, men vet inte hur man gör.
Stimuli: Sugen på sockerkaka
Respons: En färdig sockerkaka som jag bakat som jag kan stilla mitt sug med.

Indelning av problem i delproblem:
Jag måste lära mig att baka sockerkaka, vilket närmast kan hittas i en kokbok. Jag läser receptet och finner att jag behöver ingredienser och redskap. När jag har skaffat dessa kan jag börja baka min kaka.

Delproblem strukturerat:

- Lära mig baka sockerkaka
- Hitta bakredskap
- Hitta ingredienser
- Baka sockerkakan

Lösning av delproblemen:

Delproblem	Lösning
Lära baka sockerkaka	Finn en kokbok
Hitta bakredskap	Läs i kokboken och se vilka som behövs, samt titta i köket om de finns. Om de inte finns låna eller köp redskap som behövs.
Hitta ingredienser	Läs i kokboken och se vilka mängder som behövs, samt titta i köket om de finns. Om de inte finns låna eller köp ingredienser som behövs.
Baka sockerkakan	Läs i receptet vilka mängder av ingredienser som behövs och följ arbetsgången, samt grädda sockerkakan i ugnen under den tid som är föreskriven.

Lösning av huvudproblemet genom att lösa delproblemen i successiv ordning ger till slut en sockerkaka. Analyssituation har tillämpats eftersom det inte finns några givna krav på hur pass ätbar kakan skall vara. Designsituationen är mer komplex eftersom det ingår problem som har att göra med sockerkakans kvalitet och egentligen är en strukturering av nedärvd husmoderskunskap vilken inte direkt står uttalad i kokboken.

Det är också viktigt att begränsa sitt modelleringsarbete och inte försöka göra en generell modell av problemområdet. Ta t.ex. delproblemt hitta ingredienser. Om det visade sig att jag inte hade några ägg hemma skulle jag låna eller köpa nya. Det fanns inga i affären för att hönorna inte värpte, för att det varit en kall vinter, vilket beror på..... Det syns klart och tydligt att problmet baka en sockerkaka skulle svälla ut i all oändlighet om det inte begränsades till en vardaglig livssituation.

3.2 Datorrepresentation

Datorrepresentationen innebär att kunna representera den modell man arbetat fram i en datormiljö. Det är frågan om att se till att den dator som används kan förstå och lösa det problem som definierats och modellerats. Följande steg kan urskiljas:

- kunskapsrepresentation.
- sökstrategier.
- implementeringsmiljö.

3.2.1 Kunskapsrepresentaion

Beroende av det problem som är aktuellt att representera kan olika typer av kunkskapsrepresentation väljas. Några typer:

Semantiska nät: är en samling objekt som knyts samman av
länkar, samband.

Produktionssystem: regler som representerar relationer. Används ofta i rådgivande och diagnostiserande system.

Objektorienterade system: ramar. Är en beskrivning av ett objekt som innehåller specifika egenskaper, där all relevant information som kan associeras med objektet, lagras. Ställer höga krav på syntax och semantik.

Logikbaserade system: t.ex. predikatlogik, där information, kunskap om problemet beskrivs i logiska satser, axiom. Användbar för begränsade problemområden.

I praktiken finns sällan några renodlade kunskapsrepresentationer. Man måste blanda de olika typerna p.g.a. dess olika karakteristika. En typ av representation kan passa bra i ett visst delproblem men fungerar kanske sämre som representaion av ett totalt system.

3.2.2 Sökstrategier

Sökstrategin är programmets sätt att söka information i den samling kunskap som finns lagrat i det kunskapsbaserade programmet. Valet av sökstrategi beror på vilken typ av problem som föreligger.

Några typer av sökstrategier:
Framåtsökning: datadriven sökning.
Bakåtsökning: måldriven sökning.
Backtracking: t. ex. måldriven sökning i det logiska programspråket Prolog.

3.2.3 Implementeringsmiljö

Valet av implementeingsmiljö är ett val som beror på vilken typ av problem som skall lösas. Ett problem som har en rutinartad problemlösningsstruktur kan gott

och väl programmeras i ett lågnivåspråk, konventionellt språk typ Fortran eller Pascal. Ett mer komplext problemområde kräver ett språk som är mer lämpat för representation av deskriptiv kunskap, alltså ett högnivåspråk. Några typer av programspråk:

Konvetionella språk; Fortran, Pascal
Beskrivande språk; Prolog
Objektorienterade språk; Smalltalk
Skal; Insight 2
Hybrida KBS-verktyg; ART

Tillvägagångssätt hämtat från /5/

3.3 Allmänna kommentarer

Den konceptuella modelleringen är också i allmänhet ett abstrakt begrepp som är svårt att beskriva. Det beror på att människor abstraherar ett problem på olika sätt och därmed skapar individuella bilder för sig själva. Den gemensamma nämnaren är dock att skapa ordning i den dolda oredan och frilägga den implicita kunskapen.

"Anyone who has been involved in building an expertsystem knows that finding the facts and the relevant rules is the most difficult phase.The reason is that "human knowledge" to less than 1 percent is represented as explicit algorithms - most of it is of heuristic (erfarenhetsmässig) nature." / 9/.

Den konceptuella modelleringen är det första och mest grundläggande steget. Detta arbete måste utföras med största noggrannhet, då programmets prestanda och framgången med programmeringsarbetet beror på hur väl man lyckats genomskåda den bakomliggande i problemet. Datorrepresentationen av problemmodellen är mer sekundär, för den kan alltid göras om och förändras. Att förändra problemmodellen är däremot ett tidsödande arbete och ger långtgående konsekvenser för programeringsarbetet. Modelleringen kan i princip också sägas vara oberoende av datorrepresentationen , då den är generell för alla typer av datorrepresentationer. Man behöver alltså inte ta hänsyn till vilket språk som man skall programera i då man gör sin konceptuella modellering.

Det är också viktigt att begränsa sin modell och inte försöka göra den generell. Att försöka genomskåda och analysera en generell verklighet inom ett område är dömt att misslyckas, då modelleringen är avsedd att spegla de viktigaste sambanden i ett problemområde.

4 LOGISK PROGRAMMERING

För att läsaren skall ha någon behållning av rapporten måste grunderna för logisk programmering beskrivas. Om läsaren redan har kunskap inom detta område kan avsnittet hoppas över. Programspråket Prolog betyder PROGramming in LOGic och är ett av de programeringsspråk som har sin grund i logiken.

4.1 Historia

Nästan alla moderna datorer baseras på datorgurun Von Neumanns principer, vilka blev allmännt accepterade och vedertagna under 40-talet i USA. Datorn enligt Von Neumanns principer karakteriseras av ett stort likartat utrymme för minnesceller,(sekundärminne) och en processenhet med några lokala minnesceller, kallade register (primärminne). Processenheten kan lagra data från sekundärminnet eller registret, utföra aritmetiska operationer och logiska operationer med olika register, samt lagra värden från registret tillbaka till sekundärminnet. Ett program för en dator enligt Von Neumanns principer, består av en sekvens av instruktioner som skall utföras som operationer, samt en uppsättning kontrollinstruktioner som kan påverka den nästa instruktionen som skall utföras.

Allteftersom problemet med att bygga datorer och instruera dem belystes hittades också flaskhalsar. En av flaskhalsarna var männsikans oförmåga att instruera datorerna. Forskarna började därför ta fram enklare programspråk som idag har resulterat i t ex Fortran och Pascal. Dessa språk är dock märkta av det underliggande maskinspråket signerat Von Neumann. Språken är nu standard och har en vid användning, men de har ett grundläggande fel; de tvingar progarmutvecklaren att uttrycka sitt programmeringsmål i begränsade uppsättningar av operationer. Språken kan bara behandla små brottstycken av delproblemen och lösningsstrategierna måste detaljstyras in i minsta detalj. Material hämtat från /7/

Programmerandet i traditionell progamering karakteriseras mest av rutinartade styrprocesser och problem med att tala om för datorn hur den skall göra för att lösa ett givet problem. Huvudproblemets lösning i stort blir ett mycket avlägset mål då detaljkrångel ligger i dess väg.

4.2 Konceptet

Den logisk programmerigen (L.P) baserar sig på tanken att det skall vara lätt för människan att ge klara instruktioner till datorn. Logisk programmering är, till skillnad från traditionella programmeringsspråk, inte härled från Von Neumanns principer. L.P är istället härled från en abstrakt modell som inte har någon direkt relation till någon speciell maskinmodell. I L.P behöver inga utförliga instruktioner ges för hur operationerna skall genomföras för att datorn skall kunna lösa ett givet problem. Istället skall problemet beskrivas och kunskapen friläggas.
Beskrivningen skall formuleras som logiska satser,i form av satslogik eller predikatlogik, s.k axiom. Detta innebär att programutvecklaren beskriver sitt problem i en logisk kunskapsstruktur. Av denna anledning krävs ett mycket noggrant arbete med den konceptuella modelleringen. Programmet exekveras blott genom att ge en logisk fråga som programmet med sin etablerade kunskapsstruktur sätts att bevisa. Den logiska frågan är i sig problemets mål, vilket därför också blir programmets mål. Programmeraren behöver egentligen inte bekymra sig över hur programmet löser ett givet problem, utan programmeringen inriktas huvudsakligen på att beskriva problemet på ett så tydligt sätt som möjligt.

4.2.1 Det logiskt programmets arbetssätt:

Program	Beräkning	Användare
Uppsättning	Konstruktivt	Fråga
axiom	bevisande av	
	fråga.	

4.3 Fundamentala begrepp

I logisk programmering finns tre fundamentala beståndsdelar:

- Fakta
- Frågor
- Regler

4.3.1 Faktan

En fakta är ett påstående utan villkor. Påståendets grundfunktion är att etablera, fastställa, en relation mellan argument och eller egenskaper, vilka som helst, som har en viss samhörighet. Begreppet relation brukar också kallas för predi-

kat. En fakta består av ett predikat och en eller flera argument och brukar formuleras: predikat(argument1,....,argumentn)

Exempel 1.1
Påstående i naturligt språk:
En tomat är röd!
Påstående i predikatlogik:
tomat(röd); predikat: tomat
argument: röd

Exempel 2.1
Påstående i naturligt språk:
Sven har ett barn som heter Kalle
Påstående i predikatlogik:
barn(sven,kalle) predikat: barn
argument: Sven, Kalle

Det spelar ingen roll hur argumenten är placerade i relation till varandra inom predikatet. Placeringen kan man själv bestämma. Det innebär att faktan barn(sven,kalle) = barn(kalle,sven) i betydelse som fakta. Men ett program i sin enklaste form byggs upp av en faktatyp men med varierande argumentsinnebörd

Exempel 3.1
barn(Ulla,Eva)
barn(Ulla,Lotta)
barn(Erik,Eva)
barn(Erik,Lotta)

Genom definitionen barn(föräldernamn,barnnamn) har detta enkla program betydelsen att Ulla och Erik har barnen Lotta och Eva. Här har programmeraren bestämmt sig för att föräldern står för barnet i argumentlistan, annars blev innebörden att Eva och Lotta har barnen Ulla och Erik. Detta är en logisk konsekvens av programmet men det är orimligt eftersom två kvinnor inte kan få barn.

4.3.2 Frågan

En fråga är i sig ett påstående, som programmet skall bevisa. Frågan är också ett sätt att få information från programmet, men mer om det senare. Frågan frågar programmet om en viss relation finns mellan 1 till n argument. Frågan ser i lo-

gisk form lika dan ut som faktan, vilket kan vara förvillande. Men man skall komma ihåg att frågan, i sin enklaste form i yttersta ledet, ställs av användaren, vilket innebär att frågan tillförs programmet som dess mål som skall bevisas. Frågan är alltså inte en del av kunskapsstrukturen, på det enklaste planet.

Exempel 1.2
Fråga i naturligt språk:
Är en tomat röd ?
Fråga i predikatlogik: tomat(röd)?

Exempel 2.2
Fråga i naturligt språk:
Har Sven ett barn som heter Kalle ?
Fråga i predikatlogik: barn(sven,kalle)?

Här bör man observera att logiken bara ger begränsad information. Om man frågar barn(sven,kallle)? Vad får man egentligen reda på ? Man får reda på att Sven har ett barn som heter Kalle, och man kan vända på frågan och få reda på att Kalle har en förälder som heter Sven. Men L.P kan inte säga om Kalle och Sven är man eller kvinna. Allså går det inte att dra slutsatsen att Sven är fader och Kalle son. Viktigt att komma ihåg då man jobbar med logisk programmering !

När programmet bevisar en fråga är det samma sak som att bestämma om frågan är en logisk konsekvens av programmet. Biter man sig i svansen ? Nej, man skall komma ihåg att det logiska programmet är en representation av den kunskap som finns inom ett problemområde t.ex. Kalle och Sven har en familjerelation av något slag. Informationen finns i programmet och "som man frågar får man svar" heter det. Frågar man om Sven har ett barn som heter Kalle och programmet besitter den kunskapen, får man också som en logisk konsekvens av programmet, svaret sant.

4.3.3 Regler

Allmänt sett består en regel av en slutsats som kan programmet kan dra om ett eller flera villkor är uppfyllda;
Slutsats om vilkor1 ... vilkorn är uppfyllda.

Lättast att illustrera regeln är att fortsätta på förälder-barnrelationen. Tidigare har vi bara kunnat ta reda på om Sven har någar barn t.ex. Kalle. Nu är vi dock

intresserade av veta om Sven har en son som heter Kalle. För den operationen behövs fakta:
barn(sven,kalle)
man(sven)
man(kalle)

För att ta reda på om Sven har en son som heter Kalle, son(sven,kalle)? måste en regel ställas upp.
Naturligtspråk:
Om Y har ett barn X och X är man, då har Y en son X.
Predikatlogik:
son(Y,X):- barn(Y,X) och man(X).

För att kunna förklara reglernas generella uppbyggnad måste först begreppet variabler förklaras. Ta exempel 2.2 som utgångspunkt. Frågan ställdes om Sven hade ett barn som hette Kalle; barn(sven,kalle)? Om vi nu skulle vilja veta vad Svens son heter, eller vad Kalles fader heter. Hur gör man då ? Jo man ersätter namnen med variabler istället. Frågan vad Svens barn heter blir då: barn(sven,X)? där variablen X står för alla möjlig barn som Sven kan ha. Variabler inleds alltid med versaler medan alla argument- och predikatnamn inleds med gemener. Vad Kalles föräldrar heter, blir då barn(X,kalle) och vill man veta vilka föräldra-barn relationer som finns i programmet, då skriver man barn(X,Y), X- föräldernamn, Y-barnnamn.

Innebörden av en variabel i logisk programmering skiljer sig från innebörden av densamma i traditionell programmering. Variabeln i det senare fallet är en i princip betydelselös pekare till en minnespost. Variabeln har en typ t.ex. real eller integer men den har ingen specifik betydelse. Den kan representera antal potatisar, eller hjul på en bil eller antal rätt på stryktipset etc. Den logiska variabeln har däremot, trots att den inte tilldelats ett visst värde, en betydelse genom att den ingår som representant för ett argument i ett predikat. Variabeln kan ha många olika värden, men värdena har samma betydelse. Argumentet (en) presenteras av en (flera) ofyllda variabler som har en definierad relation till övriga variabler Denna regelkonstruktion är generell och är ett sätt att bygga mer komplexa relationer mellan argument och argument och relationer och relationer. Man kan delvis säga att en regel också är en fakta eftersom regeln definerar nya relationer på en högre nivå. Sett ur synvinkel konceptuell modellering är regelkonstruktionen ett sätt att bygga upp en struktur; i botten de minsta kunskapsdelarna och allt eftersom göra kunskapsstrukturen allt mer komplex.

5 EXEMPLET TOKBALK

För att i någon mån illustrera KBS-världens möjligheter och begränsningar i byggnadsindustrin, har vi konstruerat ett kunskapsbaserat program. Programmet utför dimensionering av ett rektangulärt betongtvärnsitt i brottgränstillståndet. Handböcker, erfarenhetsmässig kunskap och egenhändigt hopsnickrade tumregler har lagrats i programmet som kunskap.

Vi har valt att först presentera programmets funktion och arbetssätt. Detta för att visa att KBS-tekniken är ett effektivt och tidsbesparande verktyg i byggprocessen. När läsaren orienterat sig i programmet, följer den konceptuella modelleringen, d.v.s hur vi brutit ner problemet i dess beståndsdelar. Modelleringen följs av en utvärdering och en del synpunkter på problem som vi stött på och erfarenheter vi fått av arbetet.

5.1 Funktion

Programmet beräknar erfoderlig armering i en rektangulär betongbalk. Om så önskas väljer programmet också erfoderlig betong- och armeringskvalitet, stångdimension samt antal armeringslager. Kontroll sker av att beräknat antal stänger rymms i balken, beaktande av täckande betongskickt och minsta avstånd mellan paralella stänger. Programmet undersöker även att vald stångdimension finns till vald armeringskvalitet enligt Betonghandboken kap 3:1 tabell :2 Dimensionering sker enligt BBK-79 i brottstadium enkelarmerat tvärsnitt. Programmet väljer vid beräkning lägsta möjliga betong- respektive armeringskvalitet och minsta möjliga stångdimension, samt lägst antal lager, för att klara given påkänning. Programmet kontrollerar även om det överhuvud taget behövs någon armering.

Indata ges i form av moment från lasteffekt (Mu). Balkens bredd och höjd och önskad säkerhetsklass enligt Betonghandboken kap 1.3, samt antal lager. Dessutom ges möjlighet för användaren att välja egna värden på betongkvalitet, armeringskvalitet , stångdimension och antal armeringslager.

Resultatet från programmexekveringen presenteras i två delar, kontrollutskrift och resultatutskrift. Kontrollutskriften innehåller kvitto på given indata, bestående av:

* DIMENSIONERANDE MOMENT [kNm]
* BALKBREDD [m]
* BALKHÖJD [m]
* SÄKERHETSKLASS
* BETONGKVALITET
* STÅLKVALITET
* ANTAL LAGER [st]

De beräknade värdena presenteras i resultatsutskriften enligt två alternativ. Alternativ ett presenteras då ingen armering erfodras för att klara beräknade dragspänningar, och alternativ två presenteras då armering erfodras.

ALTERNATIV ETT	ALTERNATIV TVÅ
* "Ingen armering erfodras"	* ARMERINGSAREA [m2]
* BERÄKNAD SPÄNNING UK [MPa]	* ANTAL STÄNGER [st]
* TILLÅTEN SPÄNNING UK [MPa]	* TÄCKSKICKT [mm]

5.2 Arbetssätt

Programmet beräknar erfoderlig armering med beräkningsgången:

1. $m = Mu/B*d^2*fcc$
2. $w = 1- 1-2*m$
3. $As = w*B*d*fcc/fst$
4. $Ant = As/(Sa+0.5)$

m = relativt moment
Mu = dimensionerande moment
B = balkbredd
d = effektiv höjd
fcc = tillåten tryckspänning i betong
fst = tillåten dragspänning i armering
As = armeringsarea
Sa = stångarea

Beräkningarna följer Betonghandboken kap 3.6:43 och utnyttjar förenklat spänningsfördelning. Därmed beräknas relativ tryckzonshöjd med: $w = 0.8*X/d$.

5.3 Databasen

De material, dimensioner och antal armeringslager som är aktuella för problemet finns lagrade i databasen. Programmet konsulterar databasen då användaren inte själv definierat något eller flera av nedanstående värden.
Följande värden finns lagrade:

BTG KVALITET	ARM. KVALITET	DIAM	ANT. LAGER
K8	Ss 220	5	1
K12	Ss 260	6	2
K16	Ks 400 S	8	3
K20	Ks 400	10	4
K25	Ps 500	12	5
K30	Ks 600	16	6
K35	Ss 700 A	20	7
K40		25	8
K50		35	9
K55			10
K60			
K70			
K80			

*Beteckningar anknyter till planerade förändringar i svensk standard.
OBS! Ss 700 A finns ej mera.

Till respektive betongkvalitet följer information om:

* karakteristisk tryckhållfasthet. [fcck]
* karakteristisk draghållfasthet. [fctk]
* karakteristiskt värde på elasticitetsmodul. [Eck]

värdena är hämtade ur Betonghandboken kap 3:1 tabell :1.

Till respektive armeringskvalitet följer uppgifter om:

* till vilka dimensioner kvaliteten tillverkas.
* karakteristisk hållfasthet. [fyk]
* wbal.

De två översta uppgifterna är hämtade ur Betonghandboken kap 3:1, tabell :2 och den sista ur kap 3:6, tabell :2.

5.4 Programkörning

För att illustrera programmets funktion rent konkret har vi valt ett exempel ur Betong och Grundläggning AK I, 1983, Avdelningen för Bärande Konstruktioner, LTH, sidan 51. Exemplet kan både läsas direkt ur uppsatsen eller körs parallelt med bifogat program.

Bestäm armering för ett tvärsnitt:

Grundfakta:
Moment = 150 kNm
Bredd = 0.3 m
Höjd = 0.432 m
Säkerhetsklass 2

Materialfakta:
Betong K25
Stål Ks 400 < = 16
Diameter är valfri

```
INDATA
Dim. MOMENTET  [ kNm ] ?150
Balk-BREDD [ m ] ?      .3
Balk-HÖJDEN  [ m ] ?    .432
SÄKERHETSKLASS 1/2/3/?  2
ANTAL ARMERINGSLAGER ?  ?

SYS/ANV DEF. KVALITETER

BETONG:   SYSTEM     ANVÄNDARE
STÅL:     SYSTEM     ANVÄNDARE
DIAMETER  SYSTEM     ANVÄNDARE

MATA IN DATA < RETURN >
```

Först matas siffervärden in och programmet får köra en preliminär beräkning med kvalitetsväljarna ställda på system.

```
INDATA
Dim. MOMENTET  [ kNm ] ?150
Balk-BREDD [ m ] ?      .3
Balk-HÖJDEN  [ m ] ?    .432
SÄKERHETSKLASS 1/2/3/?  2
ANTAL ARMERINGSLAGER ?  ?

SYS/ANV DEF. KVALITETER

BETONG:   SYSTEM     ANVÄNDARE
STÅL:     SYSTEM     ANVÄNDARE
DIAMETER  SYSTEM     ANVÄNDARE

MATA IN DATA < RETURN >
```

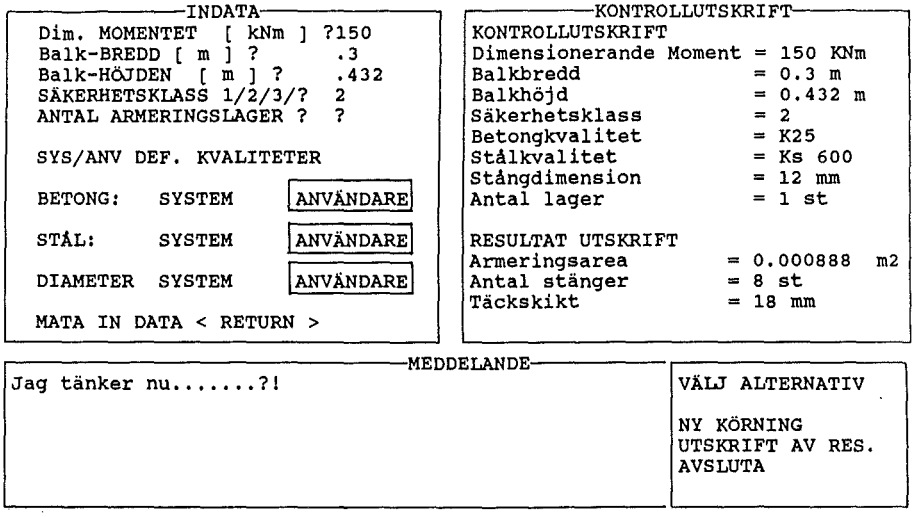
```
KONTROLLUTSKRIFT
KONTROLLUTSKRIFT
Dimensionerande Moment = 150 KNm
Balkbredd              = 0.3 m
Balkhöjd               = 0.432 m
Säkerhetsklass         = 2
Betongkvalitet         = K25
Stålkvalitet           = Ks 600
Stångdimension         = 12 mm
Antal lager            = 1 st

RESULTAT UTSKRIFT
Armeringsarea        = 0.000888  m2
Antal stänger        = 8 st
Täckskikt            = 18 mm
```

```
MEDDELANDE
Jag tänker nu.....?!
```

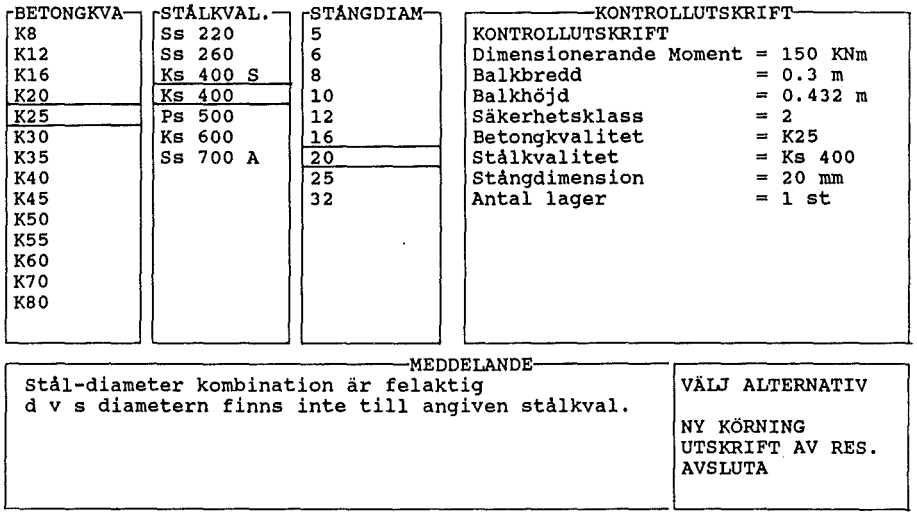
```
VÄLJ ALTERNATIV

NY KÖRNING
UTSKRIFT AV RES.
AVSLUTA
```

Programmet valde stålkvaliten till Ks 600 och det tycker du är för högt.
Du vill använda exemplets värden.

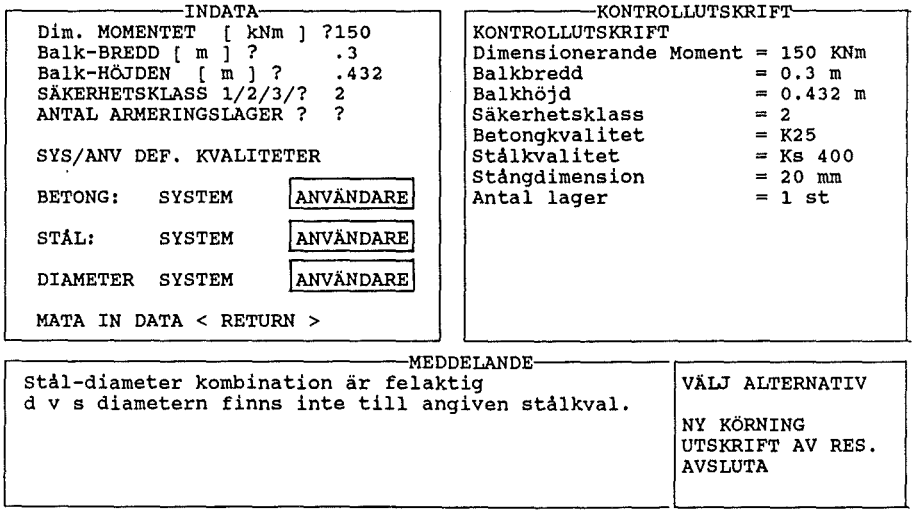
Du ställer materialväljarna på läge användare och ger vagnretur.



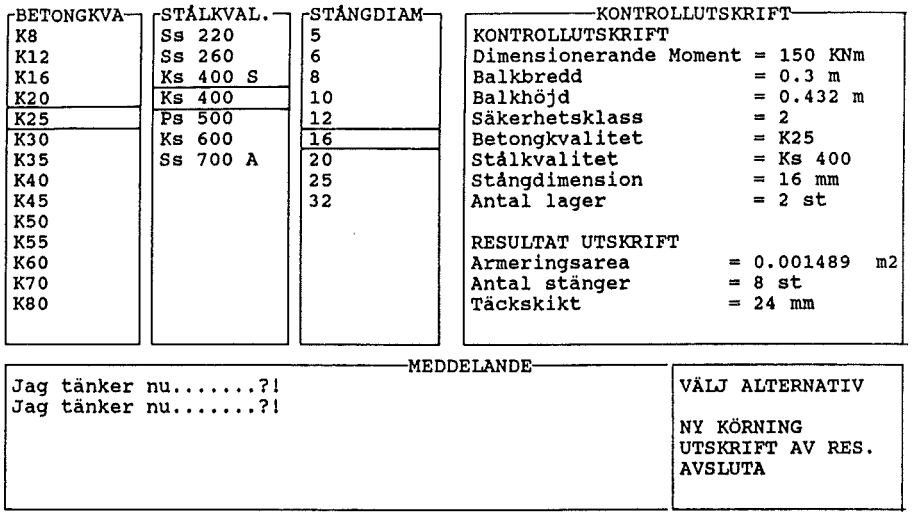
Respektive tillgänliga materialkvaliteter visas och man kan göra sitt materialval.



Du valde 20 mm armeringsstål till Ks 400 i stålkvalitet, men det visar sig att 20-stål inte tillverkas för stålkvaliteten Ks 400.



Du väljer exemplets diamter på 16 mm och erhåller som svar 8 st. järn i två lager. Du kan även få utskrift från programmet.




```
KONTROLLUTSKRIFT
Dimensionerande Moment = 150 KNm
Balkbredd              = 0.3 m
Balkhöjd               = 0.432 m
Säkerhetsklass         = 2
Betongkvalitet         = K25
Stålkvalitet           = Ks 400
Stångdimension         = 16 mm
Antal lager            = 2 st

RESULTAT UTSKRIFT
Armeringsarea      = 0.001489  m2
Antal stänger      = 8 st
Täckskikt          = 24 mm
```

I exemplet beräknades armeringsarean till 1371 m^2 medan programmet fick den samma till 1489 m^2. Skillnaden beror på att den effektiv höjden var given i exemplet medan programmet beräknar den effektiva höjden som en geometrisk tyngdpunkt beroende på det antal lager som behövs för att balken skall bli normalarmerad.

6 KONCEPTUELL MODELLERING

I detta kapitel vill vi demonstrera hur vi utförde den konceptuella modullering-en av vårt dimensioneringsproblem. Detta arbete ledde sedan till det fysiskt användbara programmet TOKBALK. Modelleringsarbetet sker enligt tidigare i tre steg:

1. Allmän formulering av problemet och indelning i delproblem.
2. Indelning i delproblem.
3. Lösning av delprobelm.

I blockstrukturen är alla delproblem isolerade och informationsflödet mellan dessa löst. Nästa steg är implementering av strukturen, vilken redovisas i bilaga 1. Av programmeringstekniska skäl måste blockstrukturen enligt ovan rivas till en del, men funktionellt sett är huvudstrukturen bibehållen.

6.1 Formulering av problemet.

Vi ville åstadkomma ett;

* datorprogramm som dimensionerar en betongbalk.
* enkelt hjälpmedel för konstruktören.
* underlättande och tidsbesparande dimensioneringsverktyg.
* datorprogram som klarar all slags indata utan att kasta ur användaren.
* datorprogram som kan köras utan manual.
* intelligent program som själv kan välja vissa materialkvalitet och dimensioner.

Begränsningar:

* endast rektangulär betongbalk.
* endast balanserad armering.
* ingen grafisk presentation.
* endast beräkning i brottgränstillstånd
* ingen dimenisonering av tvärkraft
* ingen dimensionering av dubbelarmerat tvärsnitt

Förutsättningar:

* krafter från lasteffekt givna, ingen beräkning av snittkrafter.
* säkerhetsklass anges av användaren.
* geometri (bredd och höjd) skall vara given.

Det främsta problemet vid datormässig dimensionering av en betongbalk är att beräkningen är iterativ, samt att de två materialkvaliteterna och dimensionerna på stålet väljs antingen slump- eller erfarenhetsmässigt. En mängd information (normkrav,beställarens krav, arbetsutförande) måste samlas för att kunna välja materialkvaliteter och dimensioner svarande mot given påkänning. Detta leder till många handbokskonsultationer. Processen är både omständig och tidsödande att genomföra, med hänsyn till problemets ringa omfång och komplexitet.

Beräkningen skall kunna genomföras snabbt, ha hög flexibilitet, innehålla lättillgänglig information från bl.a normer samt ha alla materialkvaliteter integrerade i beräkningen. Om konstruktören har liten erfarenhet skall denne få vägledning genom en preliminär beräkning som enkelt kan revideras. Om det skulle visa sig att kvaliteter, lasteffekt och tillgänglig geometri ej uppfyller normkraven skall konstruktören få råd och tips för att kunna lösa sitt problem.

6.2 Indelning i delproblem och lösning

Huvudproblemet har delats upp i följande fem delproblem:

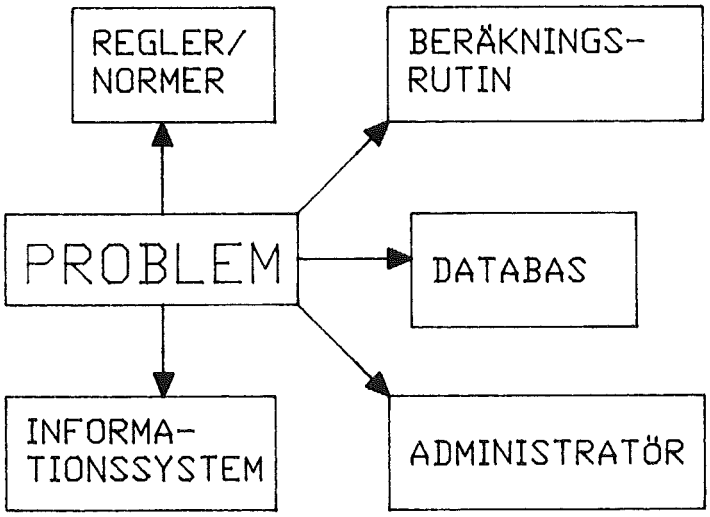


Fig 6.2 Problmet uppdelat i delproblem.

1. Regler/Normer.
2. Beräkningsrutin.
3. Databas varur programmet själv kan välja.
4. Administativt system som reglerar graden av interaktion mellan dator/användare.
5. Ett informationssytem innehållande;
   a. indatarutin.
   b. variabelbeskrivning.
   c. utdatarutin.
   d. felmeddellanderutin.

6.2.1 Regler /Normer

Informationsstyrande:

* Indata skall vara rimlig.

(Moment > 0.00, Bredd > 0.00, Höjd > 0.00, etc)

* Beräknade resultat skall vara rimliga. (Ej fler lager än där rymms i dragzon, Effektivhöjd (D) Stångdimension (Fi) * 2)
* Indata skall skrivas ut som kvitto.
* Beräknade resultat skall skrivas ut.
* Användare skall upplysas när regelbrott sker och erhålla en vägledande förklaring, vilken aktiveras:

-då alla möjliga fria variabler är uttömda.
-då någon av ovanstående är användarvald och därmed låst.
-vid direkt regelbrott t.ex. då vald diameter ej finns för önskad stålkvalitet.

Beräkningsstyrande:

* Rel.mom.48.
* WWbal.
* Betongspänning () Tillåten btg.spänning (fct).
* Då programmet själv väljer nytt Fi skall ny beräkning utföras med förändrad effektivhöjd.

Regler för utförande:

* Godkänn endast Fi som existerar till vald stålkvalitet.
* Antal lager (N) bör ej överstiga 10 st.
* Antalet armeringsstänger skall få plats i balk.
* Minsta tillåtna täckskikt är 10 mm, annars 1.5*Fi.

Allmänna regler:

* Vid nyval av variabler; (material, Fi, ant.lager)

-välj lägsta möjliga btg.kvalitet.
-välj lägsta möjliga arm.kvalitet.
-välj minsta möjliga stångdimension.
-börja med ett antal arm. lager och öka stegvis.

6.2.2 Beräkningsrutin

Beräkna enligt BBK-79, brottstadium och normalarmerad rektangulärt tvärsnitt. Alla laster är givna.

Rutin

* $\sigma = M*6/B*H^2$.
* Rel.mom = $M/B*D^2*fcc$.
* W = 1- 1-2*Rel.mom.
* Armeringsarea (As) = W*B*D*fcc/fst.
* Antal stänger (Ant) = $As*4/*D^2 + 0.5$.
* Plats i balk (b) = Ant*Fi*Konstant.

6.2.3 Databasen

Som uppfyller följande krav:

* en förteckning som innehåller data om betong, stål och stångdimensioner nödvändig för dimensionering.
* datan skall vara lagrad så att sökning sker enligt kriteriet från lägsta till högsta kvalitet/dimension (linjär sökning).
* vara konstant tillgänglig för övriga programmet.

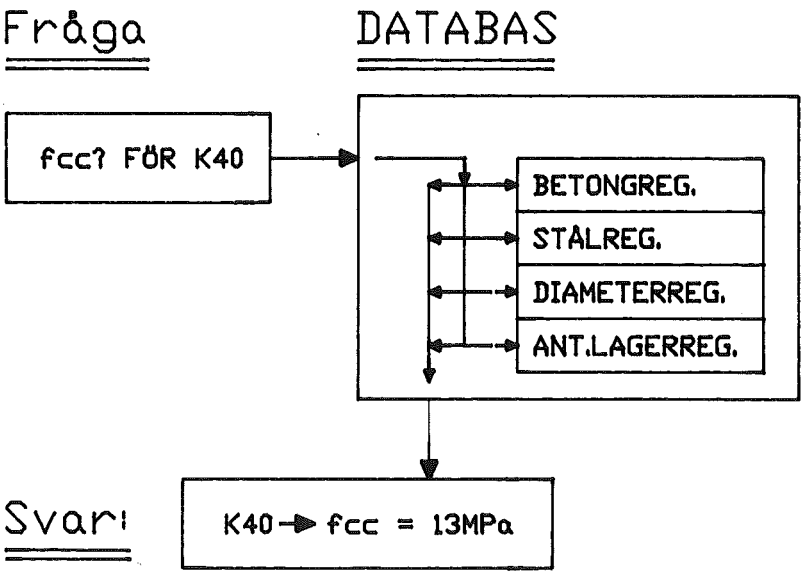


Fig. 6.2.3 Databasen konstituerar det aktuella problemområdets fakta.

6.2.4 Administrativt system

Som styr programexikveringen så att:

* beräkningen hämtar data från registren om användaren ej själv vill giva motsvarande data.
* relevant felmeddelande skrivs ut.
* resultat- och kontrollutskrift skrivs ut.
* reglerna i dess olika varianter tillfrågas vid rätt tillfälle.

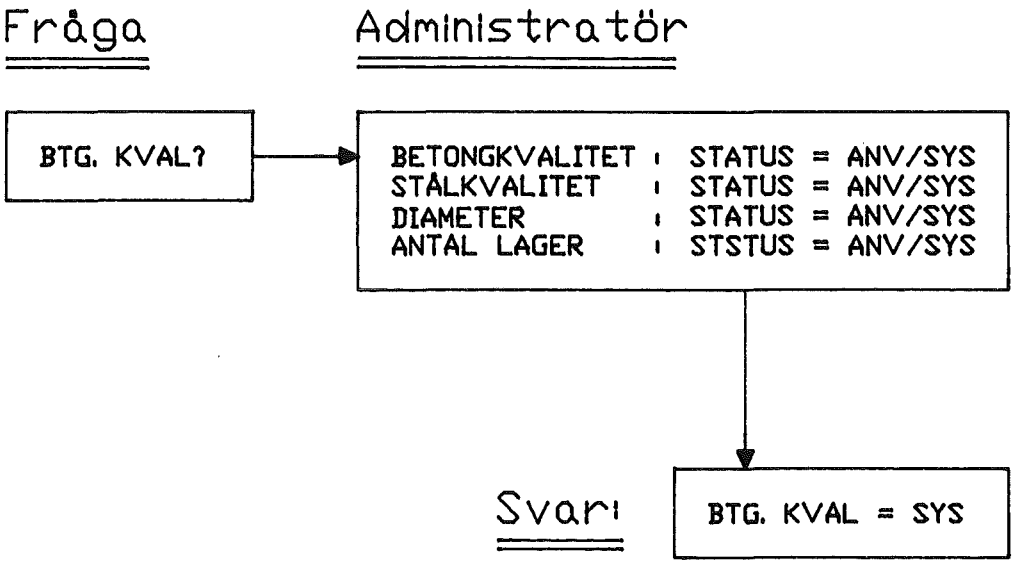


Fig. 6.2.4 Administratören reglerar informationsflödet i programmet.

6.2.5 Informationssytem

a. Indatarutin.
Indatarutinen skall sköta insamling och bearbetning av information från användare. Den skall vara lätt att använda och underlätta flera upprepade körningar. Därför skall denna rutin minnas indata från närmast föregående körning och visa dessa på en meny i vilken användaren kan ändra önskad indata.

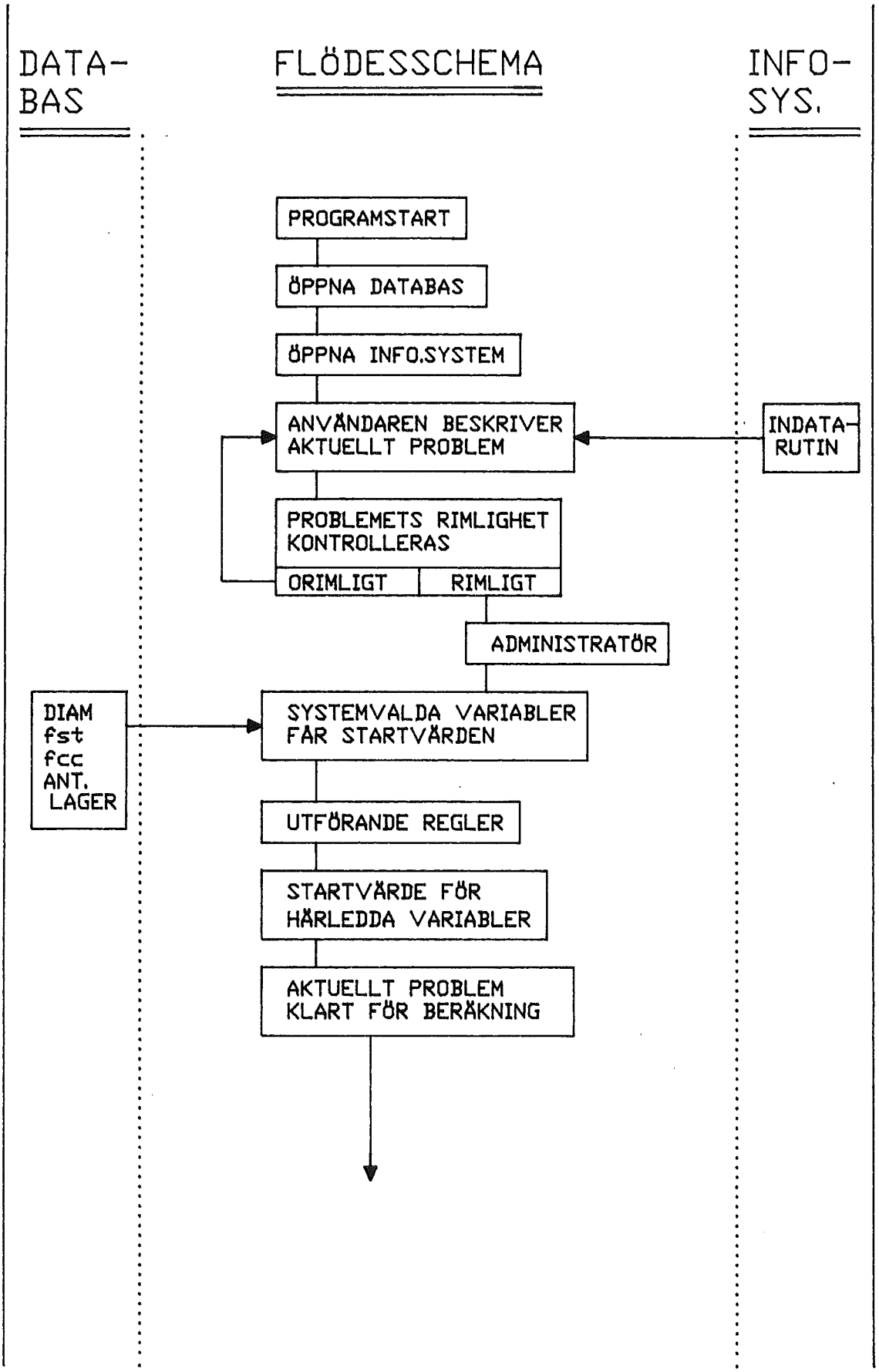
b. variabelbeskrivning.
Denna del skall förklara ingående parametrars (indata/utdata mm) för användaren.

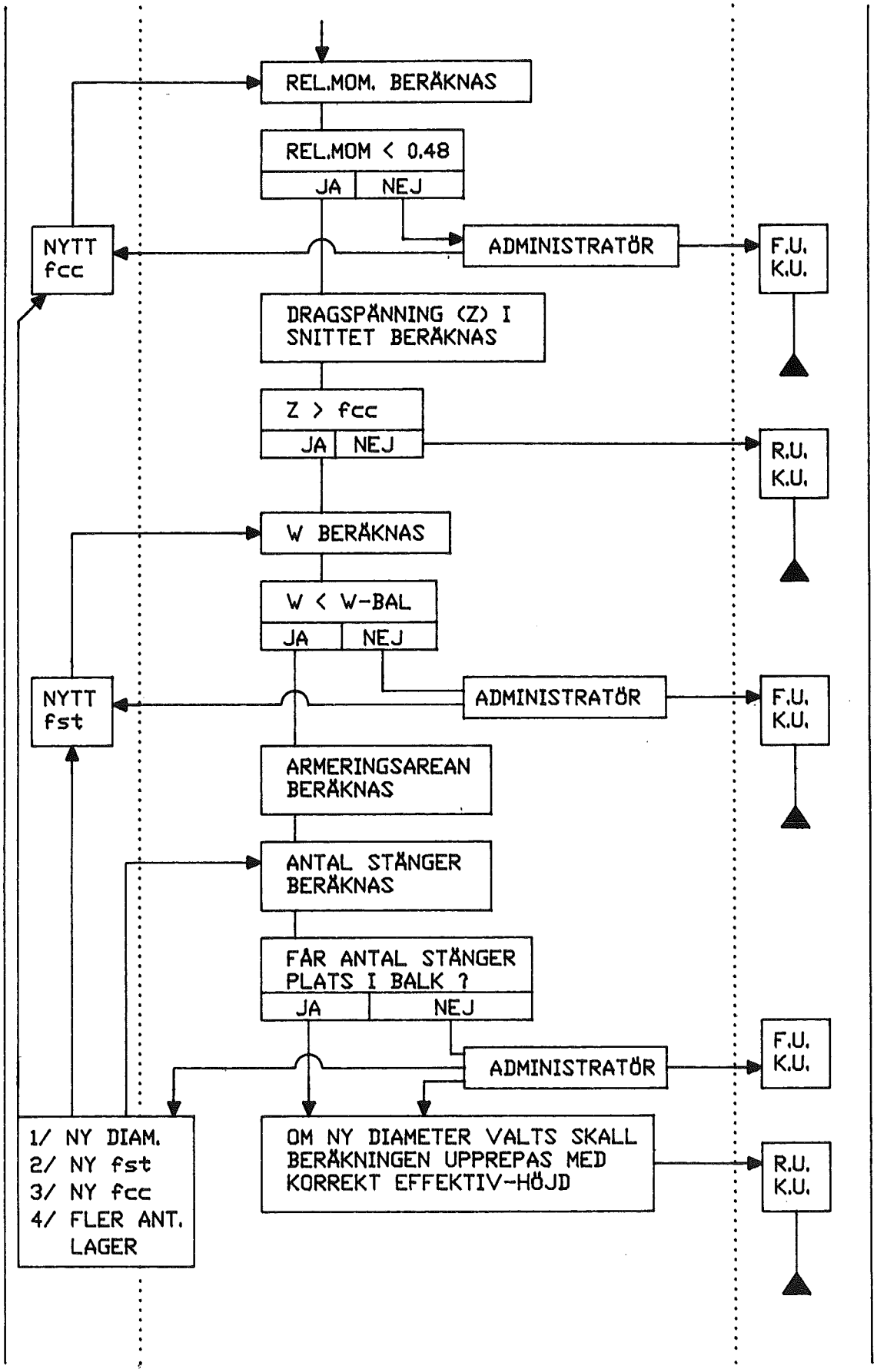
c. utdatarutin.
Innehållande:

* kontrollutskrift som ger kvitto på indata.
* resultatutskrift som överskådligt presenterar resultaten från beräkning.
* en rutin för utskrift av kontroll- och resultatuskrift på skrivare.

d. felmeddellanderutin.
Som skall uppfylla följande:

* endast visar felmeddelande som ansluter till felorsaken.
* meddelandet skall vara rådgivande och avspegla dess grund, helst med hänvisningar till litteratur inom området.
* indata i kombination med kontrollutskrift, resultatutskrift och felmeddelande skall presenteras samtidigt. Målet är att ge ett diagnoshjälpmedel med en total informationsbild av rådande problem


DATA-
BAS
FLÖDESSCHEMA
INFO-
SYS.
PROGRAMSTART
ÖPPNA DATABAS
ÖPPNA INFO.SYSTEM
ANVÄNDAREN BESKRIVER
AKTUELLT PROBLEM
INDATA-
RUTIN
PROBLEMETS RIMLIGHET
KONTROLLERAS
ORIMLIGT
RIMLIGT
ADMINISTRATÖR
DIAM
fst
fcc
ANT.
LAGER
SYSTEMVALDA VARIABLER
FÅR STARTVÄRDEN
UTFÖRANDE REGLER
STARTVÄRDE FÖR
HÄRLEDDA VARIABLER
AKTUELLT PROBLEM
KLART FÖR BERÄKNING


REL.MOM. BERÄKNAS
REL.MOM < 0.48
JA
NEJ
NYTT fcc
ADMINISTRATÖR
F.U. K.U.
DRAGSPÄNNING (Z) I SNITTET BERÄKNAS
Z > fcc
JA
NEJ
R.U. K.U.
W BERÄKNAS
W < W-BAL
JA
NEJ
NYTT fst
ADMINISTRATÖR
F.U. K.U.
ARMERINGSAREAN BERÄKNAS
ANTAL STÄNGER BERÄKNAS
FÅR ANTAL STÄNGER PLATS I BALK ?
JA
NEJ
ADMINISTRATÖR
F.U. K.U.
1/ NY DIAM.
2/ NY fst
3/ NY fcc
4/ FLER ANT. LAGER
OM NY DIAMETER VALTS SKALL BERÄKNINGEN UPPREPAS MED KORREKT EFFEKTIV-HÖJD
R.U. K.U.

7 UTVÄRDERING

7.1 Jämförelse Pascal /Prolog

Jämförelsen mellan programmering i Prolog och traditionell programmering kommer främst att göras med språket Pascal, då det är det språk vi känner bäst.

Den största skillnaden mellan Prolog och Pascal är själva grundtanken bakom respektive språk. Pascal är ett sekventiellt programspråk som genom varje funktion, subrutin och programrad specificerar hur datorn skall gå tillväga för att lösa ett visst problem. I Prolog kan man mycket väl göra ett sekventiellt program, men i grunden är Prolog ett beskrivande programspråk. Det problem som skall lösas beskrivs i tre steg och datorn har sedan fria händer att lösa problemet hur den vill. Steg:

1/ "Domains:" I denna avdelning definieras och namnges objekt som ingår i problemstrukturen.
2/ "Predicates:" Relationer mellan argument namnges.
3/ "Clauses:" Fakta och regler beskriver relationerna.

Här hämtas den aktuella kontexten och det problem som programmet ställts att lösa, härleds genom regler och fakta.

Del ett och två motsvarar Pascals deklarationsdel, medan del tre i princip är en lista av logiska satser. De logiska satserna t ex relativa m, W koll och plats i balk specificerar den önskade relationen mellan givna förutsättningar (indata) och önskat resultat (utdata).

Prolog har också en ovärderlig finess, "Backtracking." Det är en automatisk sökfunktion som aktiveras oberoende av om problemet går att lösa eller ej. Går problemet att lösa försöker Prolog att hitta alternativa lösningar som uppfyller villkoren. Går problemet inte att lösa med aktuell data, söker Prolog bakåt i programstrukturen för att försöka hitta nya värden som kan lösa problemet. Eftersom denna funktion är inbyggd och inte behöver definieras av användaren, är det mycket lätt att koppla till en databas. Databasen innehåller en mängd fakta som frigjorts från problemet. Värden som kan lösa problemet hämtas auto-

matiskt ur databasen. Dock måste små rutiner konstrueras som ger access till databasen. Förändras fakta gällande det aktuella problemområdet är det också lätt att bygga rutiner för editering av databasen. I Pascal måste programeraren själv konstruera sina sökrutiner och på rekursiv väg hitta alla lösningar till ett problem.

Pascal passar bra för implementering av matematiska samband, p.g.a. sin sekventiella natur, medan Prolog har sin styrka i att kunna representera kunskap i logiska satser. Av denna anledning kan Prolog användas för representation och problemlösning inom områden med erfarenhetsbaserad kunskap som inte går att definiera med exakta lagar och regler. Programmet TOKBALK har en kombinerad kunskaps- struktur med tumregler och matematiska samband. Det har därför varit aningen besvärligt att implementera vårt problemområde i Prologmiljö för kunskapens tyngdpunkt ligger på den matematiska sidan.

Programmerandet i Prolog har karakteriserats av en grammtisk och syntaxisk frihet. Prolog ställer stora krav på problemdeklarationen medan själva programmuppbyggnaden (clauses) har varit lättsam. Eftersom Prolog inte kräver någon speciell ordning och reda i programstrukturen har vi kunnat bygga vårt program efter hand, utan allt för stor eftertanke, och ändock fått en fungerande arbetsprototyp. Det enda krav som ställs är att varianter av samma predikat skall vara grupperade. Pascal präglas däremot av hög formalism och krav på ordningsföljd samt krav på god kännedom om språkets syntax.
Prolog är p.g.a. den enkla uppbyggnaden ett utmärkt simuleringsverktyg. En idé eller del av en problemmodullering kan snabbt implementeras för att testa samband, orsak och verkan, eller om problemet överhuvudtaget passar för implementering i Prolog. Det är allstå lätt att komma till skott i programmeringsarbetet om man programmerar i Prolog.

När den konceptuella modelleringen av ett problemområde är klart, är det också lätt att sätta upp sin huvudstruktur i implementeringen. Varje delproblem, t.ex. representerat som en box/låda i ett flödesschema/en problemstruktur, kan direkt översättas som en relation mellan argument och namnges med specifikt predikatnamn i datorprogrammet. Programmeringsarbetet kan därefter koncentreras på den interna uppbygganden av respektive predikat (delproblem). Efter hand som de programeringstekniska problemen är lösta kan de också provköras. I Pascal är det mer besvärligt att sätta sina funktioner och subrutiner eller delprogram i provbänk.

Prolog har lätt för interaktion både med program skrivna i andra språk och med användaren. Ett prologprogram kan mycket väl fungera som användarsnitt och

programskal för avancerade beräkningsprogram med bedrövliga indata- och utdatarutiner. Det är relativt lätt att skräddarsy egna menyhanteringar och editorer allt efter behov och krav på användarvänlighet.

De erfarenheter vi dragit av programmering i Prologmiljö är inte odelat positiv, men programstrukturens enkelhet och minimala krav på formalia, ger programmeringsarbetet en lätthet som vi inte upplevt med varken Pascal eller Fortran 77. Prolog har närmat sig det mänskliga tänkandet några steg. Valet av problemområde som implementeras i Prologmiljö är dock av största vikt för att få ett effektivt utnyttjande av Prologspråkets fördelar.

7.2 Tokbalks utvecklingsarbete

Det arbete vi utfört, för att konstruera ett kunskapsbaserat program för dimensionering av en betongbalk, har varit allt annat än renodlat enligt beskrivet tillvägagångssätt. I arbetets inledningsskede, när vi diskuterade storlek, mål och utformning, var problemområdet inte definerat. Däremot valde vi utan eftertanke Prolog som implementeringsmiljö. Med det valet följde måldriven linjär sökning "backtracking" som sökningsstrategi och logisk representation av kunskap.

Till en början lade vi ner stort arbete för att lära oss Prolog till fulländning. Det problemområde vi beskrivit tog vi direkt ur luften för vi tyckte det var ett enkelt problem. Föga anade vi morgondagens sanning; Prolog lämpar sig inte för problemområden definierade av matematiska samband. Att lära sig Prolog och utforma ett intelligent dimensioneringsprogram gick hand i hand. I takt med att att känndedomen om Prologspråket ökade, växte otillfredställelsen med ett oändamålsenliga programmet som levde sitt eget liv utan kontroll från konstruktörerna.

Efter ca. sex till sju veckors arbete brakade projektet samman. Vi hade ett program som själv valde materialkvaliteter och stångdimensioner, men våra intentioner att göra programmet användarvänligt med felutskrifter som gav råd till användaren vid rätt tillfälle, var ett fiasko. När momentet var för högt och beräkningarna inte kunde klara relativt moment 0.48 kom felutskrifter om att vi inte klarade normalarmerad balk etc. Vi fick bita i det sura äpplet, lägga programmeringsmanualen på hyllan, och börja göra en konceptuell modell av hur informationsflödet till användaren skulle fungera. Vi ritade beslutsträd (som omkom i samband med utflyttningen ur vårt examensaretsrum) tills vi förstod att programmet måste ha en administratör som avgjorde hur och när felmeddelanden skulle visa för användaren.

Efter lösningen av det problemt visade det sig att vi förbisett en kontrollberäkning med hänsyn till en exakt effektiv höjd. Vi valde att göra kontrollberäkningen rekursiv och då uppstod problemet med att felutskrifter visade sig under kontrollberäkningen som egentligen skulle vara stum och antingen lyckas eller misslyckas utan meddelande till användaren. Fram med penna och papper igen och vi konstaterade att kontrollberäkningen skulle placeras i ett eget predikat som skulle tillfrågas endast då kontrollberäkning skulle utföras.

På ett tidigt stadium hade vi klart för oss hur beräkningsrutinen skulle fungera och vi träffade egentligen rätt från första början. Det var informationssystemet, egentligen stödsystemet till den avskalade beräkningsrutinen, som orsakade den största huvudbryn. Bristen på modellering och definition av vad vi ville att programmet skulle prestera, skapde en oerhörd oreda i arbetet. Dokumentationen av tidigare arbete och arbetsplan för framtida problem var obefintlig. När vi stötte på ett problem värderade vi inte dess totala påverkan på systemet utan begränsade oss till ett snävt perspektiv. Det gav negativa spinnoff-effekter på programsystemet i sin helhet, vilka var svåra att hitta källan till när man väl glömt bort hur man löst ett delproblem en gång i tiden.

Känslan av misslyckande började infinna sig då programmet mer och mer likande ett sekventiellt program. "Vad är det för skillnade på vårt program och ett Pascalprogram med samma funkiton ?," var frågan som vi ofta ställde oss. En analys av hur vi gått tillväga och en del litteraturstudier visade att det probelm vi valt att lösa med logisk representation av kunskap, inte var anpassat för Prologmiljön. Problemets sekventiella natur kräver en sträng styrning av informationsflödet genom programmet. Denna förödande slutsats skulle vi ha dragit vid valet av datorrepresentaion av problemmodellen. Valet hade då stått mellan att välja en mer avpassad mjukvara eller att helt förkasta problemområdet och väla ett nytt som var avpassat för Prologmiljön. Ett normstyrt problemområde med ickematematiska samband hade varit bättre för vår del. Det hade förmodligen i större utsträckning visat fördelarna med Prolog som programmeringsspråk.

Vid en närmare titt på filfördelingschemat (bilaga 1) finner man att informationssystemet är det helt dominerande problemområdet. Vi har funderat en del kring detta och kommit fram till tre möjliga förklaringar.
1/ Informationssystemets dominans avspeglar att användarvänligheten har varit vårt verkliga problemområde och inte att göra ett intelligent konstruktionsprogram.

2/ Med anledning av att vi inte var förtrogna med programmeringsspråket Prolog i början av arbetet, har vi inte heller lyckats ta vara på Prologs naturliga förutsättningar för smidig informationshantering inom systemet och ut mot användaren.

3/ Att dimensionera en betongsbalk med avseende på böjning är inget avancerat problem ur matematisk och hållfashetsmässig synpunkt. Det svåra i problemet är att hantera informationen som krävs för dimensioneringen och därför har informationssystemet blivit det centrala problemområdet.

Att uttala sig om vilken av förklaringarna som är mest sannolik, är svårt. Vi nöjer oss med att konstatera förklaringarna dels isolerade och dels som enhet.

Programmet TOKBALK med dess samlade fördelar och nackdelar är resultatet av vårt examensarbete, rent konkret. Det mer ogreppbara resulatet av arbetet är de erfarenheter och färdigheter vi tillägnat oss under arbetets gång. Den senare delen är det område som är svårast att beskriva men som ger det mesta att ha i bagaget för framtidens yrkesverksamhet.

Arbetets huvudmål var att visa att kunskapsbaserad programvara är något att satsa på för byggnadsindustrin. I detta fall får byggnadsindustrin stå för konstruktionsprocessen som isolerad företelse.

Den mest påtagliga slutsatsen av arbetet är att framställning av kunskapsbaserad programvara kräver mycket god kännedom om de inblandade kompetensområdena. Man måste behärska konstruktionsområdet, eller åtminstone den del man är intresserad av att datorsiera, fullständigt. Samtidigt måste man vara väl förtrogen med den implementeringsmiljö som man väljer att utföra programmeringsarbetet i. Med dessa färdigheter i bagaget står man sig dock ganska slätt om man inte behärskar förmågan att formulera, frilägga, och strukturera det aktuella problemområdet. Att kräva dessa tre egenskaper ev en konstruktionsspecialist är en orimliget. För att framställa KBS-programvara krävs professionella kunskapsingenjörer som bemästrar implementeringsmiljö och systemeringskunnande, samt har förmågan att snabbt sätta sig in i specialistens problemområde och därefter utföra den konceptuella modelleringen.

Konceptuell modellering är egentligen en formalisering av den problemlösningsprocess som varje människa går igenom för att lösa ett problem. Om vi går tillbaka till problemet "Baka en sockerkaka" (sidan 13). Du är sugen på sockerkaka och vill baka den. Vad gör du ? Förmodligen tar du fram kokboken, fixar in-

gredienser och bakredskap, smäller ihop ingredienserna, sätter in kakan i ugnen, och gräddar den. Du har fått din sockerkaka och löst ditt problem. Vid tillvägagångssättet koncetuell modellering får man se problemet ur en annan synvinkel. Du vet behov, tillvägagångssätt och resultat i mer eller mindre medveten form, d.v.s det vållar dig inga större svårigheter att lösa problemet och du vet intuitivt problemets begränsingar, förutsättningar och rimlighet. Begränsningar, förutsättningar och rimlighet är begrepp som datorer inte har någon uppfattning om. Man måste därför formalisera problemet och beskriva det explicit i logiska satser och termer, och därmed bygga upp en värld som är en spegling av verkligheten. Denna värld skall sedan implementeras och därmed har datorn fått en värld att röra sig inom.

Tidigare beskrev vi att man måste begränsa sin modell och delvis där ligger pudelns kärna i programutvecklingen. Man måste klart och koncist rama in sitt problem och definiera inom vilket område man rör sig. Inom ett visst område finns mitt problem och de orsakskedjor som anknyter till problemområden utanför definierade ramar måste avskiljas och ersättas med axiom eller antaganden. Först då låter sig problemet modelleras.

8 FRAMTIDEN

8.1 Allmänt

Den tekniska utvecklingen går allt snabbare. Nya material och tekniska lösningar provas, förnyas, förändras och introduseras. Detta gäller inom alla industrigrenar inklusive byggbranschen. Våra nya elektroniska hjälpmedel blir allt billigare och därför vanligare i det dagliga arbetet. Gemensamt för dessa hjälpmedel är att de alla på något vis förmedlar information och kunskap, fig 8.1. Idag är det nödvändigt att ha tillgång till mycket information för att kunna prestera en bra lösning. I litteraturen stöter man på begreppet "informationssamhället", ett samhälle vari tillgång till riktig information är ett krav. AI-tekniken är ett hjälpmedel som både är nödvändigt och anpassat för detta samhälle, bild . I första hand direkt som en länk mellan människa och maskin, men även som lätthanterliga kunskapsbaser, biliotek, register och expertsystem för informationshantering.

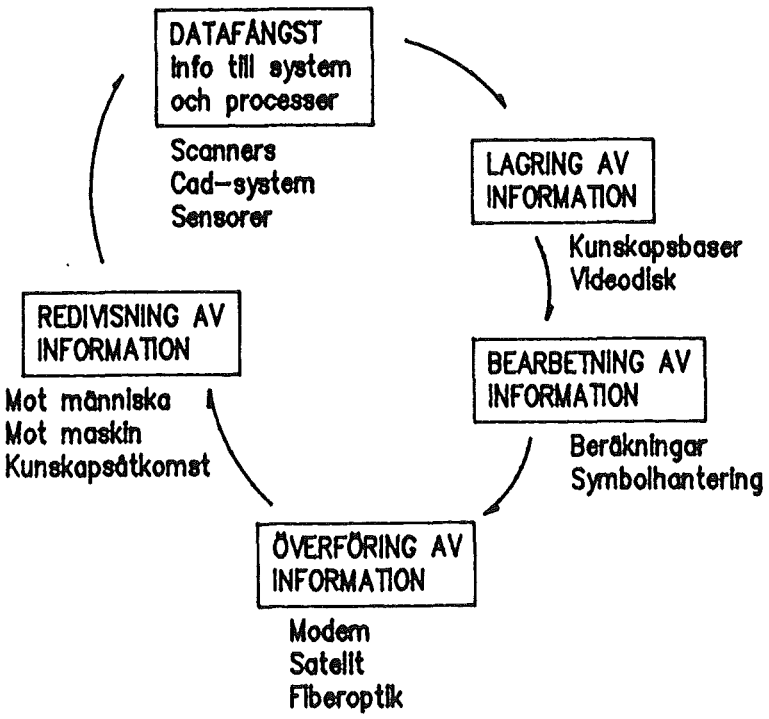


Fig. 8.1 Illustration av informationsflödet i olika nivåer i en process.

Följande fördelar kan vi vänta oss av AI-tekniken:

* integrerad mjukvara. (t.ex. koppling CAD-Beräkning)
* bättre och tillgängligare information.(t.ex. sökrutiner med sållning för databaser)
* flexiblare preprocessorer. (enklare att ge indata till t.ex. FEM-program)
* möjlighet för symbolbibliotek. (för t.ex. koppling CAD-KBS-databas)
* bättre pre- och post-prossecorer med möjlighet för bl.a. grafik.
* system som inte behöver programmeras utan kan läras att lösa ett problem.

Samtidigt måste vi akta oss för system som ger automatiska lösningar där maskinens förmåga styr människans tänkande istället för tvärt om.

8.2 Expertsystem

Den gren inom AI som vunnit störst kommersiell framgång är utvecklingen och försäljningen av expertsystem. Orsaken torde vara att argumenten för dessa är lättillgängliga. Med ett expertsystem kan företagets experter frigöras från tidskrävande rutinuppgifter och ägna sig åt mer kvalificerade uppgifter eller vidareutveckling. Systemet i sig erbjuder sedan hela företagsorganisationen ett effektivt beslutsstöd, en maskinell expert som kan vägleda i rutinmässiga frågor. Men man måste komma ihåg att utvecklingen av dessa expertsystem kan ta lång tid, under vilken just de bästa och mest belastade experterna blir hårt ansträngda.

Utvecklingen av expertsystem går allt snabbare. Detta kommer av att nya och effektivare verktyg utveklas te.x. KBS och färdiga program "skal". De nya skalen är av en typ där kunskapen ej nödvändigtvis behöver omformuleras till regler. Istället skriver man in ett antal exempel ur vilka skalet generaliserar olika regler. En typ av avlusningsmöjlighet kallad "debugging". Skalen utvecklas även mot mer användarvänlighet, med grafiska snitt vilka ger en helt annan överblick över regelbasen med mera fig 8.2.

För närvarande tillverkas mest skal skrivna i traditionella programspråk, typ C. Detta kan vara en nackdel eftersom anpassade AI-språk ,typ Lisp och Prolog, redan i sin struktur har löst en del av de tekniska problem som AI-programmering innebär. Man talar här om högnivåspråk. Dessutom kräver dessa språk mindre minnesutrymme och är lättare att integrera med externa program.

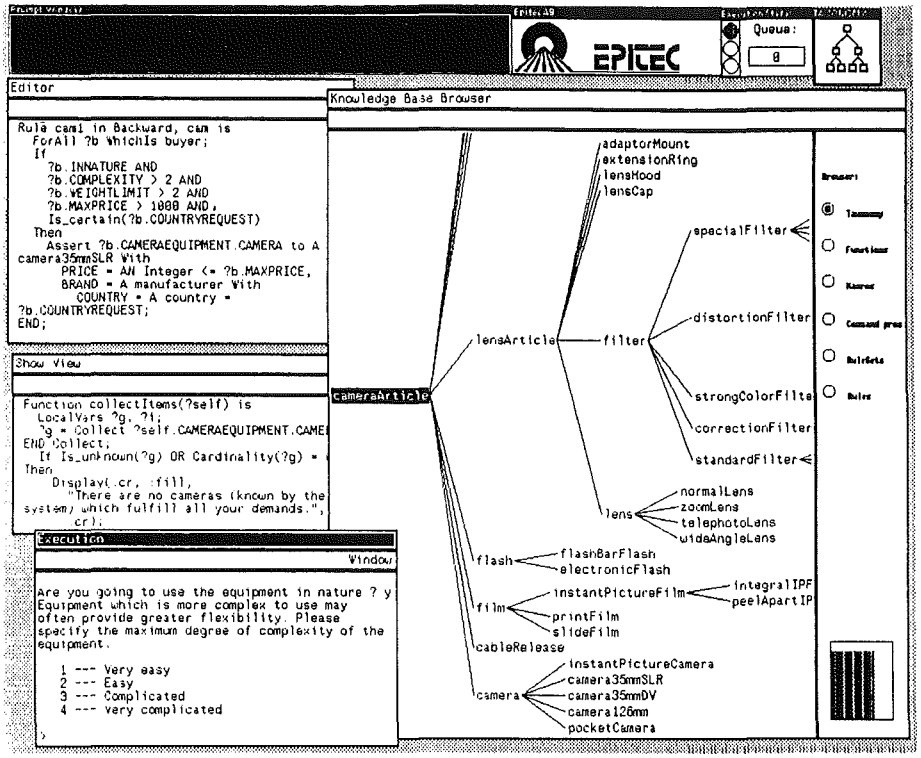


Fig. 8.1 Exempel på skal med grafiskt användarsnitt. /Epitool Knowledge Software, Linköping Sweden, 1987/

8.3 Tillämpningar

Kommersiella expertsystem började först dyka upp inom områden för diagnos och felsökning. Orsaken var bland annat att maskiner kan repareras snabbt och inte behöver stå stilla. Vilket snabbt gav ekonomiska fördelar.Billiggare mjuk och hårdvara inom datorindustrin har på senare tid ytterligare ökat spridningen inom olika områden. Här följer några exempel. /8/

Ekonomi/Finans

Kreditanalys
Skatterådgivning
Budgetering
Besluts rådgivning (management)
Riskanalys
Fösäkringsrådgivning

Teknik

CAD/CAM stöd
Materialval för tillverkningsprocesser
Utvärdering av signaler från olika sensorer

Front-end till komplexa simuleringar
Diagnos av delsystem i bilar
Felsökning i datanätverk
Testning av kretskort

Databehandling

Databas administration
Front-end till komplicerade mjukvaror

Försäljning och marknadsföring

Konfigurering av datorer
Val av delar från produktkatalog

Produktion

Diagnos och underhåll av maskiner
Verktygsval för automatisk tillverkning
Diagnos av kraftförsörjning
Processkontroll

Utbildning

Simulering av processer
Simulering för management träning
Val av litteratur

Medicin

Diagnos av EKG

LITTERATURFÖRTECKNING

1. "Artificial intelligence and natural man." Boden 1977
2. Ennals R, 1985, "AI-applications to logical reasoning and hisorical research".
3. "GEPSE a computer enviroment for engenering solving problems."
4. Val. Sillén R, "Building PC-hosted expert systems using induction methods." Novacast Expert Systems AB, Ronneby, Sverige.
5. Christiansson P Herrera A, 1987, "Kunskapsbaserade system/Expertsystem värderinng av några existerande verktyg." NBS-DATA seminarium Oslo sept 1987.
6. Christiansson P, 1987 "Datorisering av byggprocessen i Sverige. Internationellt perspektiv". Bärande konstruktioner LTH. Arbetsrapport jan 1987.
7. Sterling L, Shapiro E, 1986, "The art of Prolog."
8. Persson J, 1987, "Expertsystem i USA." Utlandsrapport från Sveriges tekniska attachéer, nov 1987.
9. Christiansson P Herrera A, 1986,"Properties of future knowlegde based systems. The interactive system example." Singapore may 1986.
10. Christiansson P, 1986. "Högteknologi i byggindustrin. Vad sker i andra länder?" 3B-programmets årskonferens nov 1986.
11. Åkerlund S, 1983, "Stål och Trä konstruktion",Bärande konstruktioner LTH. Kursbok AK II.
12. Winston, Patric, Henry, 1984,"Artificial intelligence"
13. Lemmon E J, 1965, "Beginning logic"
14. Clocksin W F, Mellish C S, 1981, "Programming in Prolog"
15. Bubenko J, Lindencrona E,1984, "Konceptuell modellering-informationsanalys"

BILAGOR

1. Filfördelningsschema

2. Programutskrift

PROBLEM-OMRÅDEN / RUTINER	REGLER/ NORMER	BERÄKNINGS-RUTIN	DATA-BAS	ADMINI-STRATÖR	INFO-SYSTEM
TOKB2.PRO	JA	JA			
TOKBALK.DBA			JA		
TOKWRITE.PRO					JA
BALKUT.BAT					JA
UTDATA.DAT					JA
TOKREAD.PRO	JA	JA			JA
TOKSAFE.PRO	JA				JA
TOKBALK.DAT					JA
TOKMAT.PRO					JA
TOKATTR.PRO				JA	
TOKEDIT.PRO					JA

```
/************************************************************
                    "TOKB2.PRO, new edition"

Program för beräkning av armering av en rektangulär betongbalk.

TILLÄMPADE REGLER
R_m < 0.48          Annars ej tillräckligt stor tryckzon. ( X < D )
W   < W_bal         Endast enkelarmerade tvärsnitt beräknas.
Dmin < Fi < Dmax    Att vald stångdiameter finns till vald armeringskva.
Balkbredd >  Antal stänger plus täckskickt.
************************************************************/

nowarnings
code=5000

project "TOKBALK"

include "TOKGLD.PRO"
include "TOKREAD.PRO"
include "TOKEDIT.PRO"
include "TOKWRITE.PRO"

predicates
   run
   start(indatatyp,armtyp,betongtyp,fityp,kontrolltyp)
   relativa_m(indatatyp,armtyp,betongtyp,fityp,real,kontrolltyp)
   cal_R_m(ask_symb,ask_symb,real,real,real)
   enbart_btg(indatatyp,armtyp,betongtyp,fityp,real,real,kontrolltyp)
   w_koll(indatatyp,armtyp,betongtyp,fityp,real,real,kontrolltyp)
   omega(real,real)
   cal_Z(ask_symb,ask_symb,ask_symb,real)
   arm_area(indatatyp,armtyp,betongtyp,real,real,real)
   plats_i_balk(indatatyp,armtyp,betongtyp,fityp,real,kontrolltyp)
   fi_koll(indatatyp,armtyp,betongtyp,fityp,real,kontrolltyp)
   cal_T_skikt(fityp,real)
   cal_koll(indatatyp,armtyp,betongtyp,fityp)
   aprove_N(real,ask_symb,lagertyp,fityp,real)
   heltal(real,fityp)
   ok_nyN(armtyp,betongtyp,kontrolltyp)

/*** Deklaration av predikat som ingår i huvudrutin TokB2.pro ***/

goal
   set_windows,
   consult("tokbalk.dba"),
   shiftwindow(9),
   presentation,
   shiftwindow(10),
   material_write,
   data_write,
   rerun,
   run,nl,
   choose.

/*** Programmets mål som det skall uppfylla. ***/
```

```
clauses /* KÖRNINGSDEL */

   run:-
      shiftwindow(2),
      data_editor,
      uni_ask(indata,Data,Kont),
      Kont = kontroll(Fcc_val,Fst_val,Diam_val,_),
      läs_fcc(Data,Btg,Fcc_val),
      läs_fst(Data,Arm,Fst_val),
      läs_diam(Fi,Diam_val),!, /* får ej fråga användare en gång till */
      shiftwindow(6),
      clearwindow,
      start(Data,Arm,Btg,Fi,Kont),!.

/*** läs_X- kommandon förser system med data från användaren. ***/

   start(Data,Arm,Btg,Fi,Kont):-
      write("Jag tänker nu.......?!"),nl,
      Kont = kontroll(Fcc_val,Fst_val,Diam_val,N_val),
      fråga_diam(Fi,Diam_val),
      fråga_N(Data,N_val),
      fråga_eff_höjd(Data,D,Fi),
      fråga_fcc(Data,Btg,Fcc_val),
      relativa_m(Data,Arm,Btg,Fi,D,Kont).

/*** fråga_X - kommandon förser systemet med data från databaser. ***/

CLAUSES  /* BERÄKNINGSDEL */

   relativa_m(Data,Arm,Btg,Fi,D,_):-
      D< Fi*2*0.0010,
      K=kontroll("Dneg","Dneg",_,_),
      fel_utskrift(Data,Arm,Btg,Fi,K).
   relativa_m(Data,Arm,Btg,Fi,D,Kont):-
      Data=indata(Moment,Bredd,_,_,_),
      Btg=btg(_,Fcc,_,_),
      cal_R_m(Moment,Bredd,D,Fcc,R_m),
      R_m <= 0.48,
      enbart_btg(Data,Arm,Btg,Fi,D,R_m,Kont).
   relativa_m(Data,Arm,Btg,Fi,_,Kont):-
      Kont = kontroll(Fcc_val,_,_,_),
      Fcc_val = "anv",
      K=kontroll("anv","R_m",_,_),
      fel_utskrift(Data,Arm,Btg,Fi,K).
   relativa_m(Data,Arm,Btg,Fi,_,_):-
      Btg=btg(Kval,_,_,_),
      Kval = "K80",
      K=kontroll("sys","R_m",_,_),
      fel_utskrift(Data,Arm,Btg,Fi,K).

/*** Predikatet relativa_m kontrollerar att tryckzonen inte blir större än **
/*** effektiv höjd. De olika versionerna av relativa_m finns för att ge fel-
/*** utskrifter (vägledning) för respektive flexibilitetsgrad. ***/
```

```
    enbart_btg(Data,Arm,Btg,Fi,D,_,Kont):-
       Data=indata(M,H,B,_,_),
       Btg=btg(_,_,Fct,_),
       cal_Z(M,B,H,Z),
       Fct>=Z,
       Arm=arm("INGEN",_,_,_,_),
       kontroll_utskrift(Data,Arm,Btg,Fi),
       resultats_utskrift(Z,Fct,0).
    enbart_btg(Data,Arm,Btg,Fi,D,R_m,Kont):-
       Kont = kontroll(Fcc_val,Fst_val,Diam_val,_),
       omega(R_m,W),
       w_koll(Data,Arm,Btg,Fi,D,W,Kont).

/*** Predikatet enbart_btg kontrollerar om tvärsnittet eventuellt kan klara *
/*** sig utan armering. ***/

    w_koll(Data,Arm,Btg,Fi,D,W,Kont):-
       Kont = kontroll(_,Fst_val,_,_),
       Fst_val = "anv",
       Arm=arm(_,_,_,_,W_bal),
       W <= W_bal,
       arm_area(Data,Arm,Btg,D,W,As),
       fi_koll(Data,Arm,Btg,Fi,As,Kont).
    w_koll(Data,Arm,Btg,Fi,D,W,Kont):-
       Kont = kontroll(_,Fst_val,_,_),
       Fst_val = "sys",
       W<=0.616,
       fråga_fst(Data,Arm,Fst_val),
       Arm=arm(_,_,_,_,W_bal),
       W <= W_bal,
       arm_area(Data,Arm,Btg,D,W,As),
       fi_koll(Data,Arm,Btg,Fi,As,Kont).
  w_koll(Data,Arm,Btg,Fi,_,_,Kont):-
       Kont = kontroll(Fcc_val,Fst_val,_,_),
       Fcc_val = "anv",Fst_val="anv",
       K=kontroll("anv","anv","W_bal",_),
       fel_utskrift(Data,Arm,Btg,Fi,K).
  w_koll(Data,Arm,Btg,Fi,_,_,Kont):-
       Kont = kontroll(Fcc_val,Fst_val,_,_),
       Fcc_val = "anv",Fst_val = "sys",
       K=kontroll("anv","sys","W_bal",_),
       fel_utskrift(Data,Arm,Btg,Fi,K).
  w_koll(Data,Arm,Btg,Fi,_,_,Kont):-
       Kont = kontroll(Fcc_val,Fst_val,_,_),
       Fcc_val = "sys",Fst_val = "sys",
       Btg=btg(Bkval,_,_,_),
       Bkval="K80",
       K=kontroll("sys","sys","W_bal","sys"),
       fel_utskrift(Data,Arm,Btg,Fi,K).
  w_koll(Data,Arm,Btg,Fi,_,_,Kont):-
       Kont = kontroll(Fcc_val,Fst_val,_,_),
       Fcc_val = "sys" , Fst_val = "anv",
       Btg=btg(Bkval,_,_,_),
       Bkval="K80",
       K=kontroll("sys","anv","W_bal",_),
       fel_utskrift(Data,Arm,Btg,Fi,K).

/*** Predikatet w_koll kontrollerar att balken blir normalarmerad. De olika
/*** versionerna ger felutskrifter (vägledning) för de olika felxibilitets  *
/*** graderna.  ***/
```

```
rerun.
rerun:-rerun.

cal_R_m(Moment,Bredd,D,Fcc,R_m):-
   R_m=Moment*0.001/(Bredd*Fcc*D*D).

omega(R_m,W):-
   W=1-sqrt(1-2*R_m).

cal_Z(M,B,H,Z):-
   Z=(M*6)/(B*H*H*1000).

cal_T_skikt(Fi,T_skikt):-
   Fi >= 10,
   T_skikt = 1.5*Fi.
cal_T_skikt(Fi,T_skikt):-
   Fi < 10,
   T_skikt = 15.

arm_area(Data,Arm,Btg,D,W,As):-
   Data=indata(_,Bredd,_,_,_),
   Btg=btg(_,Fcc,_,_),
   Arm=arm(_,_,_,Fst,_),
   As=W*Bredd*D*Fcc/Fst.

heltal(Real,Int):-
   Int=Real.

/*** Predikaten ovan är enbart beräkningsrutiner, som ***/
/*** förser regelpredikat med beräknade värden. ***/

   fi_koll(Data,Arm,Btg,Fi,As,Kont):-
      Kont=kontroll(_,Fst_val,Diam_val,_),
      Fst_val="anv",Diam_val="anv",
      Arm=arm(_,Dmin,_,_,_),
      Fi < Dmin,
      K=kontroll("anv","anv","finns_ej",_),
      fel_utskrift(Data,Arm,Btg,Fi,K);
      Kont=kontroll(_,Fst_val,Diam_val,_),
      Fst_val="anv",Diam_val="anv",
      Arm=arm(_,_,Dmax,_,_),
      Fi > Dmax,
      K=kontroll("anv","anv","finns_ej",_),
      fel_utskrift(Data,Arm,Btg,Fi,K).
   fi_koll(Data,Arm,Btg,Fi,As,Kont):-
      plats_i_balk(Data,Arm,Btg,Fi,As,Kont).

/*** Predikatet fi_koll kontrollerar om användaren definierat en korrekt ***/
/*** kombination mellan stålkvalitet och diamter. T ex produceras i stål-  **
/*** kvaliteten Ks 400 S endast armeringsjärn mellan 16 Och 32 mm i diameter
```

```
plats_i_balk(Data,Arm,Btg,Fi,As,Kont):-
   Arm=arm(_,Dmin,Dmax,_,_),
   Data = indata(_,Bredd,_,_,N),
   Fi >= Dmin,
   Fi <= Dmax,
   StångA = 0.001*0.001*Fi*Fi*3.14/4,
   NyAnt = As/StångA + 0.5,
   heltal(NyAnt,Ant),
   N*Bredd>=Ant*Fi*0.001+(Ant-1)*2*Fi*0.001+2*1.5*Fi*0.001,
   kontroll_utskrift(Data,Arm,Btg,Fi),
   cal_T_skikt(Fi,T_skikt),
   resultats_utskrift(As,T_skikt,Ant).
plats_i_balk(Data,Arm,Btg,Fi,As,Kont):-
   Kont = kontroll(_,_,Diam_val,_),
   Diam_val = "sys",
   Data = indata(_,Bredd,_,_,N),
   Arm=arm(_,Dmin,Dmax,_,_),
   diam(NyFi),
   NyFi>Fi,
   NyFi >= Dmin,
   NyFi <= Dmax,
   StångA = 0.001*0.001*NyFi*NyFi*3.14/4,
   NyAnt = As/StångA + 0.5,
   heltal(NyAnt,Ant),
   N*Bredd>=Ant*NyFi*0.001+(Ant-1)*2*NyFi*0.001+2*1.5*NyFi*0.001,
   cal_koll(Data,Arm,Btg,NyFi).
plats_i_balk(Data,Arm,Btg,Fi,As,Kont):-
   Kont = kontroll(_,_,_,N_val),
   N_val = "sys",
   ok_nyN(Arm,Btg,Kont),
   Data = indata(M,Bredd,H,Sk,N),
   NyN=N+1,NyN<=10,
   NyData = indata(M,Bredd,H,Sk,NyN),
   Start(NyData,Arm,Btg,Fi,Kont).
plats_i_balk(Data,Arm,Btg,Fi,_,Kont):-
   fel_utskrift(Data,Arm,Btg,Fi,Kont).

/*** Predikatet plats_i_balk är hjärnan i huvudrutinen. Den kontrollerar slut
/*** giltitgt om framräknad betong-stål-diamter-antal lager-kombination får
/*** plats i balken. Om materialuppsättningen inte får plats görs beräkningen
/*** om och nya värden väljs från databas i den mån flexibiliteten tillåter.

ok_nyN(Arm,Btg,Kont):-
   Kont = kontroll(Fcc_val,Fst_val,_,_),
         Fcc_val="anv",
         Fst_val="anv".
ok_nyN(Arm,Btg,Kont):-
   Kont = kontroll(Fcc_val,Fst_val,_,_),
         Fcc_val="sys",
         Fst_val="sys",
         Btg=btg(Bkval,_,_,_),
         Bkval="K80";
   Kont = kontroll(Fcc_val,Fst_val,_,_),
         Fcc_val="sys",
         Fst_val="sys",
         Arm=arm(Akval,_,_,_,_),
         Akval="Ss 700 A".
ok_nyN(Arm,Btg,Kont):-
   Kont = kontroll(Fcc_val,Fst_val,_,_),
         Fcc_val="anv",
         Fst_val="sys",
         Arm=arm(Akval,_,_,_,_),
         Akval="Ss 700 A".
```

```
ok_nyN(Arm,Btg,Kont):-
   Kont = kontroll(Fcc_val,Fst_val,_,_),
          Fcc_val="sys",
          Fst_val="anv",
          Btg=btg(Bkval,_,_,_),
          Bkval="K80".

/*** Predikatet ok nyN kontrollerar om det är dags att öka antalet lager***/

    cal_koll(Data,Arm,Btg,NyFi):-
      write("Jag tänker nu.......?!"),nl,
      fråga_eff_höjd(Data,D,NyFi),
      D>NyFi*2*0.001,
      Data=indata(Moment,Bredd,H,_,N),
      Btg=btg(_,Fcc,_,_),
      cal_R_m(Moment,Bredd,D,Fcc,R_m),
      R_m <= 0.48,
      omega(R_m,W),
      Arm=arm(_,_,_,_,W_bal),
      W <= W_bal,
      arm_area(Data,Arm,Btg,D,W,As),
      StångA = 0.001*0.001*NyFi*NyFi*3.14/4,
      NyAnt = As/StångA + 0.5,
      heltal(NyAnt,Ant),
      N*Bredd>=Ant*NyFi*0.001+(Ant-1)*2*NyFi*0.001+2*1.5*NyFi*0.001,
      aprove_N(D,H,N,NyFi,W),
      kontroll_utskrift(Data,Arm,Btg,NyFi),
      cal_T_skikt(NyFi,T_skikt),
      resultats_utskrift(As,T_skikt,Ant).

/*** Predikatet cal_koll gör en kontrollberäkning av materialkombinationen  *
/*** om diamtern är systemvald. Om beräkningen håller ja då blir det resul-
/*** tatutskrift.  ***/

    aprove_N(D,H,N,NyFi,W):-
      X=W*D/0.8,
      X>H-N*2.5*NyFi.
    aprove_N(_,_,_,_,_):-
      write("Allt för många lager. Armering i dragzonen"),nl.

/*** Predikatet aprove_N kontrollerar så att armering inte ligger i tryck-  *
/*** zonen där den inte gör någon nytta.  ***/



/************* SLUT PÅ BERÄKNINGSDEL **********************/
```

```
/********** FRÅGEPROGRAM - GENERELLT *********************/

        /**********************************/
        /* symbolic file name: tokread.pro */
        /**********************************/

        INCLUDE  "toksafe.pro"
        INCLUDE  "tokmat.pro"
        INCLUDE  "tokattr.pro"

        DOMAINS
            ask_type = symbol
            lista=string*

        PREDICATES
            gådå
            set_windows
            uni_ask(ask_type,indatatyp,kontrolltyp)
            fråga_fcc(indatatyp,betongtyp,valtyp)
            läs_fcc(indatatyp,betongtyp,valtyp)
            fråga_fst(indatatyp,armtyp,valtyp)
            läs_fst(indatatyp,armtyp,valtyp)
            fråga_diam(fityp,valtyp)
            läs_diam(fityp,valtyp)
            fråga_eff_höjd(indatatyp,real,fityp)
            material_write
            data_write
            presentation
            string_word(string,lista)
            str_string(lista,string)
            str_symbol(string,symbol)
            seak(integer,integer)
            nonext(lista)
            N_check(string,lagertyp,valtyp)
            fråga_N(indatatyp,valtyp)

        CLAUSES
            gådå.
            gådå:-gådå.
            set_windows:-
                        makewindow(1,87,7,"",17,58,8,22),
                        makewindow(2,23,7,"INDATA",0,4,17,36),
                        makewindow(3,48,7,"BETONGKVAL.",0,4,17,12),
                        makewindow(4,48,7,"STÅLKVAL.",0,16,17,12),
                        makewindow(5,48,7,"STÅNGDIAM.",0,28,17,12),
                        makewindow(6,64,7,"MEDDELANDE",17,4,8,76),
                        makewindow(7,14,7,"KONTROLLUTSKRIFT",0,41,17,39),
                        makewindow(8,15,7,"",5,15,15,50),
                        makewindow(9,0,0,"VÄLKOMMEN,WELCOME,WILKAMM,BUENOS DIA
                        makewindow(10,0,0,"",0,0,25,80).

             /*** Predikatet set_windows skapar alla fönster   ***/


             /****************** INITSIALDATA FÖR REKT. BTG,BALK ***********/

             uni_ask(indata,indata(M,B,H,Klass,N),Kontroll):-
                        cursor(0,26),field_str(0,26,5,SM),str_real(SM,M),
                        cursor(1,26),field_str(1,26,5,SB),str_real(SB,B),B>0.0
                        cursor(2,26),field_str(2,26,5,SH),str_real(SH,H),H>0.0
                        cursor(3,26),field_str(3,26,1,STyp),str_char(STyp,Typ)
                        klass(Typ,Klass),
                        cursor(4,26),field_str(4,26,2,SN),N_check(SN,N,Trans),
```

```
                read_val(Kontroll,Trans).

        CLAUSES
         presentation:-

                        shiftwindow(8),
                        nl,
                        write("                  TOKIS BALKPROGRAM       "),nl
                        write("                    version 1.00  "),nl,nl,
                        write("            Skapat av KALLE TRULSSON  "),nl,
                        write("                       THOMAS BJÖRKLUND    "),n
                        write("                   Lund 1988-05-09    "),nl,nl,
                        write("                         FöR   "),nl,
                        write("       Institutionen för Bärande Konstruktione
                        write("                LUNDS UNIVERSITET"),nl,nl,
                        write("                 Tryck på någon knapp......"),
                        readchar(T).

/*** Rutin för presentation av programmet..... ***/

         data_write:-
                    shiftwindow(2),clearwindow,
                    write("  Dim. MOMENTET  [ kNm ] ?"),nl,
                    field_attr(0,26,5,112),
                    write("  Balk-BREDD [ m ] ?"),nl,
                    field_attr(1,26,5,112),
                    write("  Balk-HÖJDEN  [ m ] ?"),nl,
                    field_attr(2,26,5,112),
                    write("  SÄKERHETSKLASS 1/2/3/?  ?"),nl,
                    field_attr(3,26,1,112),
                    write("  ANTAL ARMERINGSLAGER ?  ?"),nl,nl,
                    field_attr(4,26,2,112),
                    write("  SYS/ANV DEF. KVALITETER"),
                    field_attr(6,2,23,112),
                    cursor(8,2),
                    write("BETONG:"),
                    skriv(8),
                    cursor(10,2),
                    write("STÅL:"),
                    skriv(10),
                    cursor(12,2),
                    write("DIAMETER"),
                    skriv(12),
                    cursor(14,2),
                    write("MATA IN DATA < RETURN >"),
                    field_attr(14,2,25,112).


         material_write:-
               shiftwindow(3),
               clearwindow,
               concrete_fact(X,_,_,_),
               write(X),nl,
               fail.
         material_write:-
               field_attr(0,0,10,112),
               shiftwindow(4),
               clearwindow,
               steel_fact(X,_,_,_,_),
               write(X),nl,
               fail.
```

```
material_write:-
      field_attr(0,0,10,112),
      shiftwindow(5),
      clearwindow,
      diam(X),
      write(X),nl,
      fail.
material_write:-
      field_attr(0,0,10,112).

/*** Predikaten data_write och material_write laddar indatafönster med  ***/
/*** med nödvändig valdata och menylayout  ***/

N_check(SN,_,Trans):-
      SN="? ",
      Trans="sys".
N_check(SN,N,Trans):-
      str_int(SN,N),N>0.00,
      Trans="anv".

/*** Predikatet N_check  svarar på frågan om N är system- eller användarvald.


nonext([H|T]):-
      H=" ".

string_word("",[]):-!.
string_word(Styp,[H|T]):-
      frontstr(1,Styp,H,S),
      string_word(S,T).

str_string([N|T],""):-
      N=" ",nonext(T),!.
str_string([H|T],Type):-
      str_string(T,S),
      concat(H,S,Type).

str_symbol(Str,Symb):-
      Str=Symb.

seak(R1,R):-
      cursor(R1,0),
      field_attr(R1,0,10,Att),
      Att=112,R=R1,!.
seak(R1,R):-
      R2=R1+1,
      seak(R2,R).

/*** Ovanstående predikat gör om en fönstervariabel till en symbol som  ***/
/*** kan användas för sökning i databasen.  ***/
```

```
/******************** FRÅGAR OM BETONGKVALITET ********************/

fråga_fcc(_,_,"anv").
fråga_fcc(Data,btg(Kval,Fcc,Fct,Ec),"sys"):-
      concrete_fact(Kval,Fcck,Fctk,Eck),
      Data=indata(_,_,_,Klass,_),
      Fcc=Fcck/(1.5*Klass),
      Fct=Fctk/(1.5*Klass),
      Ec=Eck/(1.2*Klass).

 läs_fcc(Data,Btg,"anv"):-
      gådå,
      shiftwindow(3),
      material_edit,
      R1=0,
      seak(R1,R),
      field_str(R,0,10,Styp),
      string_word(Styp,Lista),
      str_string(Lista,Type),
      str_symbol(Type,Typ),
      fråga_fcc(Data,btg(Typ,Fcc,Fct,Ec),"sys"),
      Btg=btg(Typ,Fcc,Fct,Ec),shiftwindow(6).
 läs_fcc(_,_,"sys").

/******************** FRÅGAR OM STÅLKVALITET ******************/

fråga_fst(_,_,"anv").
fråga_fst(Data,Arm,"sys"):-
      steel_fact(Kval,Dmin,Dmax,Fyk,W_bal),
      Data=indata(_,_,_,Klass,_),
      Fst=Fyk/(1.1*Klass),
      Arm=arm(Kval,Dmin,Dmax,Fst,W_bal),
      shiftwindow(6).

läs_fst(Data,Arm,"anv"):-
      gådå,
      shiftwindow(4),
      material_edit,
      R1=0,
      seak(R1,R),
      field_str(R,0,10,Styp),
      string_word(Styp,Lista),
      str_string(Lista,Type),
      str_symbol(Type,Typ),
      fråga_fst(Data,arm(Typ,Dmin,Dmax,Fyk,W_bal),"sys"),
      Arm=arm(Typ,Dmin,Dmax,Fyk,W_bal).
läs_fst(_,_,"sys").

/******************** FRÅGAR OM DIAMETER ********************/

fråga_diam(_,"anv").
fråga_diam(Fi,"sys"):-
     diam(Fi).

läs_diam(Fi,"anv"):-
      gådå,
      shiftwindow(5),
      material_edit,
      R1=0,
      seak(R1,R),
      field_str(R,0,10,Styp),
      str_int(Styp,Fi).
läs_diam(_,"sys").
```

```
/*************** FRÅGOR OM EFFELTIV HÖJD **********************/

  fråga_eff_höjd(Data,D,Fi):-
        Fi>=10,
        Data=indata(_,_,Höjd,_,N),
        D=Höjd-(2*Fi+(N-1)*1.25*Fi)*0.001,!.
  fråga_eff_höjd(Data,D,Fi):-
        Data=indata(_,_,Höjd,_,N),
        D=Höjd-(15+0.5*Fi+(N-1)*1.25*Fi)*0.001.

  /*************** FRÅGOR OM LAGER **********************/

  fråga_N(_,"anv").
  fråga_N(indata(_,_,_,_,P),"sys"):-
        lager(P).

  /*************** SLUT PÅ FRÅGEDEL **********************/
```

```
/*************************************************************/

/*                      TOKEDIT.PRO                          */
/*                  editerar i indatarutan                   */
/*************************************************************/

PREDICATES
    key_code1(ref,ref,ref)
    print(ref,ref,ref)
    koll(ref)
    data_editor

CLAUSES
    kör.
    kör:-kör.
    data_editor:- /* predikatet data_editor sköter editeringen av indatafönst
      cursor(0,26),
      kör,
      readchar(T),
      char_int(T,Num),
      cursor(Rad,Koll),
      key_code1(Num,Rad,Koll),Num = 13,Rad=14.

/*****************************************/

     key_code1(8,R,C):-  /* raderar ett tecken */
        R>=0,R<=5,
        C>=27,C<=36,
        C1=C-1,
        scr_char(R,C1,' '),
        cursor(R,C1),!.
     key_code1(13,14,_):-!. /* slut på exekvering av data_edit */
     key_code1(72,R,C):-  /* ett steg upp */
       R>0,
       R<=5,
       C>=26,C<=30,
       R1=R-1,
       cursor(R1,C),!.
     key_code1(N,R,C):- /* ett steg ner */
        koll(N),
        R>=0,R<4,
        C>25,C<=35,
        R1=R+1,
        cursor(R1,26),!.
     key_code1(72,R,C):- /* två steg upp */
        R=8,
        C>=12,C<=35,
        cursor(4,26),!.
     key_code1(72,R,C):- /* två steg upp */
        R>8,
        R<=14,
        C>=12,C<=35,
        R1=R-2,
        cursor(R1,12),!.
     key_code1(N,R,C):- /* två steg ner */
        koll(N),
        R=4,
        C>=12,C<=35,
        cursor(8,12),!.
```

```
key_code1(N,R,C):- /* två steg ner */
   koll(N),
   R>=8,
   R<12,
   C>=12,C<=35,
   R1=R+2,
   cursor(R1,12),!.
 key_code1(N,R,C):- /* två steg ner */
   koll(N),
   R=12,
   C>=12,C<=35,
   cursor(14,26),!.
 key_code1(77,R,12):- /* markörfält för användare eller system */
   R>=8,
   R<=12,
   field_attr(R,12,9,23),
   field_attr(R,23,9,112),
   cursor(R,23),!.
 key_code1(75,R,23):-  /* markörfält för användare eller system */
   R>=8,
   R<=12,
   field_attr(R,23,9,23),
   field_attr(R,12,9,112),
   cursor(R,12),!.
 key_code1(77,R,C):- /* markören ett steg till höger */
   R>=0,R<=5,
   C<33,
   C1=C+1,
   cursor(R,C1),!.
 key_code1(75,R,C):-  /* markören ett steg till vänster */
   R>=0,R<=5,
   C>=27,
   C1=C-1,
   cursor(R,C1),!.
 key_code1(81,R,C):-  /* predikat som flyttar markör från något läge til
   cursor(14,26),!.
 key_code1(N,R,C):-          /* skriver */
   N>=48,N<=57,print(N,R,C);
   N=46,print(N,R,C);
   N=63,print(N,R,C).

print(N,R,C):-  /* predikat som utför skrivandet */
   R>=0,R<=5,
   C>=25,C<=30,
   char_int(Te,N),
   scr_char(R,C,Te),
   C1=C+1,
   cursor(R,C1),!.

koll(N):-
   N=13;N=80.
```

```
/*************************************************************
                    "TOKWRITE.PRO"
                    /* UTSKRIFT */
*************************************************************/

PREDICATES
   message(lagertyp,armtyp,betongtyp,fityp,kontrolltyp)
   mowe(integer,integer)
   roll
   utskrift(integer)

CLAUSES
   kontroll_utskrift(Data,Arm,Btg,Fi):-
      shiftwindow(7),
      clearwindow,
      Data=indata(Moment,Bredd,Höjd,Klass,N),
      Btg=btg(B_kval,_,_,_),
      Arm=arm(S_kval,_,_,_,_),
      klass(Shklass,Klass),!,
      write("KONTROLLUTSKRIFT"),nl,
      write("Dimensionerande Moment = ",Moment," KNm"),nl,
      write("Balkbredd              = ",Bredd," m"),nl,
      write("Balkhöjd               = ",Höjd," m"),nl,
      write("Säkerhetsklass         = ",Shklass),nl,
      write("Betongkvalitet         = ",B_kval),nl,
      write("Stålkvalitet           = ",S_kval),nl,
      write("Stångdimension         = ",Fi," mm"),nl,
      write("Antal lager            = ",N," st"),nl.

   resultats_utskrift(Z,Ztill,0):-
      nl,write("RESULTAT UTSKRIFT"),nl,
      write("INGEN ARMERING ERFODERAS "),nl,
      write("Beräknad spänning UK = "),
      writef("%6.4 ",Z),write(" Mpa"),nl,
      write("Tillåten spänning UK = "),
      writef("%6.4 ",Ztill),write(" Mpa").
   resultats_utskrift(As,T_skikt,Ant):-
      nl,write("RESULTAT UTSKRIFT"),nl,
      write("Armeringsarea          = "),
      writef("%8.6 ",As),write(" m2"),nl,
      write("Antal stänger          = ",Ant," st"),nl,
      write("Täckskikt              = ",T_skikt," mm").

/*************************************************************
kontrollutskrift ges oberoende om körningen lyckats eller ej,
medan resultatutskrift ges då körningen lyckats. Kontroll-
utskriften kommer alltid i kombination med med ett message
som ger en viss vägledning till användaren hur han skall
lösa sitt problem. Message kan också vara rena varnings-
texter typ att tryckzonen överstiger den effektiva höjden.
Message är kontrollerat enligt kriteriet: alla möjligheter
skall vara uttömda innan message visas.
*************************************************************/

/************** SLUT PÅ UTSKRIFTSDEL *************/
```

```
fel_utskrift(D,A,B,F,K):-
   shiftwindow(6),
   clearwindow,
   D=indata(_,_,_,_,N),
   message(N,A,B,F,K),
   beep,
   kontroll_utskrift(D,A,B,F).

message(_,_,_,_,kontroll(_,"anv","finns_ej",_)):-
   write(" Stål-diameter kombination är felaktig"),nl,
   write(" d v s diametern finns inte till angiven stålkval."),nl,!.
message(_,A,B,F,kontroll("sys","sys","sys",_)):-
   A=arm(Akval,_,_,_,_),
   Akval="Ss 700 A",
   B=btg(Bkval,_,_,_),
   Bkval="K80",
   F=25,
   write(" Systemet har uttömt alla möjliga material-"),nl,
   write(" kombinationer för att lösa ditt problem."),nl,
   write(" Du måste minska moment, eller öka balk-"),nl,
   write(" dimension."),nl,!.
message(_,_,_,_,kontroll("anv","anv","anv",_)):-
   write("Du har angivit en för klen materialkombina-"),nl,
   write("tion, eller för litet tvärnsitt, eller"),nl,
   write("för högt moment."),nl,!.
message(_,_,B,_,kontroll("sys","anv","anv",_)):-
   B=btg(Bkval,_,_,_),
   Bkval="K80",
   write("Stål- och diameterval är för klent tilltaget."),nl,
   write("Öka materialkvaliteten eller öka tvärsnittets mått."),nl,!.
message(_,_,B,F,kontroll("sys","anv","sys",_)):-
   B=btg(Bkval,_,_,_),
   Bkval="K80",
   write("Öka din stålkvalitet eller öka tvärsnittets"),nl,
   write("mått."),nl,!.
message(_,_,_,F,kontroll("anv","anv","sys",_)):-
   write("Du måste öka betong- och/eller stål"),nl,
   write("kvalitet."),nl,!.
message(_,A,_,F,kontroll("anv","sys","sys",_)):-
   A=arm(Akval,_,_,_,_),
   Akval="Ss 700 A",
   F=25,
   write("Du har valt en för låg betongkvalitet."),nl,!.
message(_,A,_,_,kontroll("anv","sys","anv",_)):-
   A=arm(Akval,_,_,_,_),
   Akval="Ss 700 A",
   write("Du måste öka betongkvalitet och/eller dimension"),nl,
   write("på armeringsjärnen."),nl,!.
message(_,A,B,_,kontroll("sys","sys","anv",_)):-
   B=btg(Bkval,_,_,_),
   Bkval="K80",
   A=arm(Akval,_,_,_,_),
   Akval="Ss 700 A",
   write("Du ha för klen dimension på armerings"),nl,
   write("järnen. Öka den."),nl,!.
message(N,A,B,_,kontroll("sys","sys","W_bal","sys")):-
   B=btg(Bkval,_,_,_),
   Bkval="K80",
   A=arm(Akval,_,_,_,_),
   Akval="Ss 700 A",
   N=10,
   write("Går ej att lösa med normalarmerad"),nl,
   write("balk. Minska moment eller öka balkhöjd."),nl,!.
```

```
message(_,_,_,_,kontroll("anv","anv","W_bal",_)):-
   write("Du måste öka betongkvaliteten"),nl,
   write("eller minska stålkvaliteten,"),nl,
   write("för att klara normalarmerad balk"),nl,!.
message(_,A,_,_,kontroll("anv","sys","W_bal",_)):-
   A=arm(Akval,_,_,_,_),
   Akval="Ss 700 A",
   write("Du måste öka betongkvaliteten"),nl,
   write("för att klara normalarmerad balk"),nl,!.
message(_,_,B,_,kontroll("sys","anv","W_bal",_)):-
   B=btg(Bkval,_,_,_),
   Bkval="K80",
   write("Du måste minska stålkvaliteten"),nl,
   write("eller minska momentet,för att"),nl,
   write("klara normalarmerad balk"),nl,!.
message(_,_,_,_,kontroll("anv","R_m",_,_)):-
   write("Du får tryck i hela tvärsnittet.( X > D)"),nl,
   write("Välj ,om möjligt, högre betongkvalitet, "),nl,
   write("eller mindre antal lager. "),nl,
   write("Ändra annars övrig indata."),nl,!.
message(_,_,B,_,kontroll("sys","R_m",_,_)):-
   B=btg(Bkval,_,_,_),
   Bkval="K80",
   write("Du får tryck i hela tvärsnittet.( X > D)"),nl,
   write("Välj ,om möjligt mindre antal lager. "),nl,
   write("Öka bredd och höjd på tvärsnittet"),nl,
   write("eller minska momentet."),nl,!.
message(_,_,_,_,kontroll("Dneg","Dneg",_,_)):-
   write("För många lager!! Du får ingen tryckzon"),nl,
   write("Minska antalet lager alltså"),nl,!.

choose:- /* Asks which window you want */
   shiftwindow(1),
   clearwindow,
   write("VÄLJ ALTERNATIV"),nl,nl,
   write("NY KÖRNING"),nl,
   write("UTSKRIFT AV RES."),nl,
   write("AVSLUTA"),nl,
   roll,
   cursor(A,B),
   utskrift(A),
   A=4,
   shiftwindow(10).

roll:-   /* Moves white field in window */
   field_attr(2,0,20,112),
   cursor(2,0),
   rerun,
   cursor(A,B),
   readchar(T),
   char_int(T,Num),
   mowe(Num,A),
   Num=13,!.

mowe(80,R1):-
   R2=R1+1,
   R2<5,
   field_attr(R1,0,20,87),
   field_attr(R2,0,20,112),
   cursor(R2,0),!.
```

```
mowe(72,R1):-
   R2=R1-1,
   R2>1,
   field_attr(R1,0,20,87),
   field_attr(R2,0,20,112),
   cursor(R2,0),!.
mowe(13,_):-!.

utskrift(2):-fail.
utskrift(3):-
   shiftwindow(7),
   window_str(Utskrift),
   openwrite(utskrift,"utdata.dat"),
   writedevice(utskrift),
   write(Utskrift),
   closefile(utskrift),
   shiftwindow(10),
   system("balkut.bat"),
   system(""),
   fail,!.
utskrift(4).
```

```
/*****************************************************

                "TOKMAT.PRO"

    MATERIALEDITOR
    Symbolic_file_name: tokmat.pro
    Denna programrutin används för att editera de
    indatafönster som fälls ner, när  betong-
    eller stålkvalitet eller diamter är användar-
    vald (a).
*****************************************************/

   DOMAINS
      ref = reference integer
   PREDICATES
      material_edit
      kör
      move(ref,ref)
      find(ref,ref)
   CLAUSES
      material_edit:-
        cursor(0,0),
        R1=0,
        find(R1,P),
        cursor(P,0),
        kör,
        readchar(T),
        char_int(T,Num),
        cursor(R,_),
        move(R,Num),
        Num=13.

      move(R,72):-
        R>0,
        R1=R-1,
        field_attr(R,0,10,48),
        field_attr(R1,0,10,112),
        cursor(R1,0),!.
      move(R,80):-
        R<14,
        R1=R+1,
        field_attr(R,0,10,48),
        field_attr(R1,0,10,112),
        cursor(R1,0),!.
      move(_,13).

      find(R1,R):-
        cursor(R1,0),
        field_attr(R1,0,10,Att),
        Att=112,R=R1,!.
      find(R1,R):-
        R2=R1+1,
        find(R2,R).
```

```
      /*********************************************************/
      /*    Symbolic_file_name: TOKATTR.PRO                    */
      /*********************************************************/

DOMAINS
  valtyp = string

PREDICATES
  konsult(integer,valtyp)
  read_val(kontrolltyp,valtyp)
  skriv(integer)
  convert(valtyp,valtyp)
CLAUSES

  skriv(Rad):-
    cursor(Rad,12),
    write("SYSTEM     ANVÄNDARE"),
    field_attr(Rad,12,9,112).
/*---------------------------------*/
  convert(Val,Type):-
    Val = "ANVÄNDARE",
    Type = "anv".
  convert(Val,Type):-
    Val = "SYSTEM   ",
    Type = "sys".
/*---------------------------------*/
  konsult(R,Typ):-
    field_attr(R,12,9,Färg),
    Färg=112,
    field_str(8,12,9,Typ),!.
  konsult(R,Typ):-
    field_attr(R,23,9,Färg),
    Färg=112,
    field_str(R,23,9,Typ),!.
/*---------------------------------*/
  read_val(kontroll(Fcc_val,Fst_val,Dim_val,N_val),Trans):-
    cursor(8,12),
    konsult(8,Val),
    convert(Val,Fcc_val),

    cursor(10,12),
    konsult(10,Ful),
    convert(Ful,Fst_val),

    cursor(12,12),
    konsult(12,Kul),
    convert(Kul,Dim_val),

    N_val=Trans.
```

```
/******** DELPROGRAM SOM FRÅGAR EFTER SÄKERHETSKLASS *******/

/*********************************/
/* Symbolic_file_name: TOKSAFE.PRO */
/*********************************/

predicates
   window
   klass(char,real)
   secure_ask(real)

clauses
    window:-
        makewindow(1,32,4,"SÄKERHETSKLASSER",0,4,20,71).

    klass('1',1.0).
    klass('2',1.1).
    klass('3',1.2).
    klass('?',Klass):-
        window,
        clearwindow,
        file_str("tokbalk.dat",Klassdata),
        window_str(Klassdata),
        scr_attr(1,18,143),
        scr_attr(1,39,143),
        scr_attr(1,60,143),
        field_attr(15,2,21,143),
        scr_attr(15,27,143),
        cursor(15,27),
        readchar(X),
        klass(X,Klass),
        removewindow,
        shiftwindow(2),
        cursor(3,26),
        write(X),
        scr_attr(3,26,112).

    secure_ask(Klass):-
       write("  ANGE SÄKERHETSKLASS [1/2/3/?]"),
       readchar(Ch),
       klass(Ch,Klass).
```

```
*********************************************
  FIL MED NAMNET: "TOKBALK.DAT"
*********************************************
```

SÄKERHETSKLASS: 1	SÄKERHETSKLASS: 2	SÄKERHETSKLASS: 3
takplåt	(takås)	takbalk
väggplåt	bjälklag	ytterpelare
(takås)	(inre pelare)	ytterbalk
väggregel	(inre balk)	grundplatta u. yt-
golv på mark	(inre grundplatta)	terpelare
sockelbalk		(inre pelare)
		(inre balk)
		(inre grundplatta)

Byggnadsdelar inom parantes kan alternativt klassas i två säkerhetsklasser.

VÄLJ SÄKERHETSKLASS !

```
/*********************************************
              "TOKGLD.PRO"
       Används för att deklaration av  globala
       variabeltyper och predikat samt defini-
       tion av databasen.
*********************************************/
     global domains
        file = destination;utskrift
        ask_symb  = real
        betongtyp = reference btg(symbol,real,real,real)
        armtyp = reference arm(symbol,real,real,real,real)
        indatatyp = indata(ask_symb,ask_symb,ask_symb,real,lagertyp)
        fityp = reference integer
        kontrolltyp = reference kontroll(valtyp,valtyp,valtyp,valtyp)
        lagertyp = reference integer

     database
        concrete_fact(symbol,real,real,real)
        steel_fact(symbol,real,real,real,real)
        diam(fityp)
        lager(lagertyp)

     global predicates
        rerun
        choose
        kontroll_utskrift(indatatyp,armtyp,betongtyp,fityp) - (i,i,i,i)
        resultats_utskrift(real,real,fityp) - (i,i,i)
        fel_utskrift(indatatyp,armtyp,betongtyp,fityp,kontrolltyp) - (i,i,i,i
```

```
*****************************************
FIL MED NAMNET: "TOKBALK.DBA"
*****************************************

concrete_fact("K8",5.5,0.75,23)
concrete_fact("K12",8.5,0.9,24.5)
concrete_fact("K16",11.5,1.05,25.5)
concrete_fact("K20",14.5,1.2,27)
concrete_fact("K25",18,1.4,28.5)
concrete_fact("K30",21.5,1.6,30)
concrete_fact("K35",25,1.8,31)
concrete_fact("K40",28.5,1.95,32)
concrete_fact("K45",32,2.1,33)
concrete_fact("K50",35.5,2.25,34)
concrete_fact("K55",39,2.4,35)
concrete_fact("K60",42.5,2.5,36)
concrete_fact("K70",49.5,2.5,37.5)
concrete_fact("K80",56.5,2.5,38.5)
steel_fact("Ss 220",6,32,220,0.616)
steel_fact("Ss 260",6,32,260,0.591)
steel_fact("Ks 400 S",16,32,380,0.527)
steel_fact("Ks 400",6,16,400,0.518)
steel_fact("Ps 500",5,12,500,0.476)
steel_fact("Ks 600",6,16,600,0.44)
steel_fact("Ss 700 A",12,25,700,0.41)
diam(5)
diam(6)
diam(8)
diam(10)
diam(12)
diam(16)
diam(20)
diam(25)
diam(32)
lager(1)
lager(2)
lager(3)
lager(4)
lager(5)
lager(6)
lager(7)
lager(8)
lager(9)
lager(10)

*************************************************
FIL MED NAMNET: "BALKUT.BAT"
*************************************************

ECHO OFF
ECHO   UTDATA FRÅN BERÄKNING FINNS PÅ FILEN
ECHO            utdata.dat
ECHO   ANVÄND EGEN UTSKRIFTSEKVENS !
```