Montanus: Ergo er Morlille en Steen.

SCANLOG

-et DANSK LOGIK-PROGRAMMERINGS-SPROG

Kurt Fleckner & Jan Rubæk Pedersen

SCANLOG

- et dansk logik-programmerings-sprog

AF
KURT FLECKNER
&
JAN RUBÆK PEDERSEN

1. udgave, 1. oplag 1985.
ISBN 87-7260-005-5
Sats: Times, SatsSystem, Århus.
Tryk: AKA-Print, Århus.

I tilknytning til bogen er der udgivet en diskette med programeksempler og øvelser fra bogen.

Programdiskette og SCANLOG - et Dansk Logik-Programmerings-Sprog kan bestilles ved henvendelse til:
Forlaget SFU ApS, Klokkerfaldet 88,
8210 Århus V. Tlf. (06) 15 83 40.

Indholdsfortegnelse

1. DEL - INDFØRING I SCANLOG

Forord

Siden 1970 er flere og flere mennesker blevet interesseret i logik-programmerings-sprog. I den periode blev de første logik-programmerings-sprog skabt og anvendt på relativt store datamaskiner. Senere har man til mikro-datamater udviklet et logik-programmerings-sprog, som blandt andet anvendes til edb-undervisning af børn i England. I denne bog skal vi se på et dansk udviklet logik-programmerings-sprog, SCANLOG.

Man kan spørge, om det virkelig er nødvendigt med endnu et logik-programmerings-sprog, og til det spørgsmål vil vi svare, Ja! Vi mener, at logik-programmerings-sprog er en spændende og anderledes måde at lære at bruge datamater på og vil derfor gerne være med til at udbrede kendskabet til logik-programmerings-sprog. De andre sprog »taler« engelsk, og de stammer fra en periode, hvor man ikke tænkte så meget på, at det er mennesker, der skal anvende dem. Kort sagt så er de andre sprog ikke særligt brugervenlige, og jo sværere det er at bruge et produkt, jo færre kan eller vil bruge det. Derfor har vi udviklet SCANLOG. SCANLOG er for det første langt mere brugervenligt end de andre logik-programmerings-sprog, og for det andet »taler« SCANLOG dansk. SCANLOG udmærker sig desuden ved at indeholde et redigeringssystem, hvor alle kommandoer aktiveres ved tryk på en enkelt tast, og SCANLOG kan derpå forklare, hvordan en løsning til et problem er fundet.

Hvad er logik-programmerings-sprog så for noget? De mere almindelige programmerings-sprog, såsom BASIC, COMAL og PASCAL, er sprog, hvori man skridt for skridt beskriver, hvad datamaskinen skal lave. Man giver altså en nøje tilrettelagt plan, som datamaskinen følger. I logik-programmerings-sprog beskriver man, hvilke regler datamaskinen skal følge under løsning af et problem. Logik-programmerne er en beskrivelse af spillereglerne for løsning af et problem, mens almindelige programmer er en opskrift på, hvordan problemerne skal løses.

Bogen består af 2 dele. 1. del er en indføring i logik-programmering i SCANLOG og indeholder mange eksempler på logik-programmer. 2. del af bogen er et opslagsværk, hvor man kan finde ud af, hvad de enkelte ting betyder, og hvad man kan lave med SCANLOG.

De første 3 kapitler indeholder en beskrivelse af logik-programmerings-sproget SCANLOG samt mange eksempler for at lette forståelsen. Eksemplerne kan anvendes senere, når man selv programmerer. Kapitlerne er delt op i en gennemgang af SCANLOG baseret på eksempler, og en kørselsvejledning som kan bruges, når man ønsker at afprøve eksemplerne. Kapitlernes 1. del kan således læses uden at bruge en maskine. Samtidigt gives der en grundig og letfattelig introduktion til logik-programmering. Kørselsvejledningen forudsætter, at læseren sidder ved en datamaskine, og den indeholder eksemplerne fra kapitlernes første del samt en forklaring på, hvordan eksemplerne afprøves på datamaskinen. I kapitlernes sidste del findes opgaver af såvel teknisk- som anvendelsesorienteret art. Løsninger til opgaverne findes bagest i bogen.

I kapitel 4 fortæller vi lidt om, hvordan SCANLOG virker, og hvad man kalder de forskellige ting. Vi ser også på risikoen for at lave fejl i programmer og slutter af med at lave et program, som f.eks. kan finde vej gennem en labyrint. Kapitlet kræver forhåndskendskab til logik-programmering. Nybegyndere bør derfor først læse kapitel 1 til 3.
Kapitel 5 indeholder nogle illustrerende eksempler, som alle ligger på SCANLOG-disketten. Vi skal blandt andet se eksempler på bøjning af franske verber, kodning af tekst, protein-syntese, oversættelse af tidsangivelser fra dansk til engelsk, simplificering af aritmetiske udtryk og symbolsk differentiation.
SCANLOG er udviklet af undertegnede ved Datalogisk Afdeling (DAIMI) på Århus Universitet, under vejledning af Mogens Nielsen (DAIMI) og Michael Madsen (ekstern lektor ved DAIMI), som begge har bidraget med mange ideer og konstruktiv kritik til såvel SCANLOG som denne bog. Vi takker Regnecentralen for udlån af datamaskine.

Den 25. juli 1985.
Kurt Fleckner & Jan Rubæk Pedersen.

1. DEL - INDFØRING I SCANLOG

1. Et indledende eksempel

1.1 Fakta

Det første eksempel handler om en del af den danske kongefamilie. I eksemplet beskriver vi data om kongefamilien på en måde, som SCANLOG kan forstå. Den form for data, vi først skal se på, kaldes for *fakta*. Et eksempel herpå er

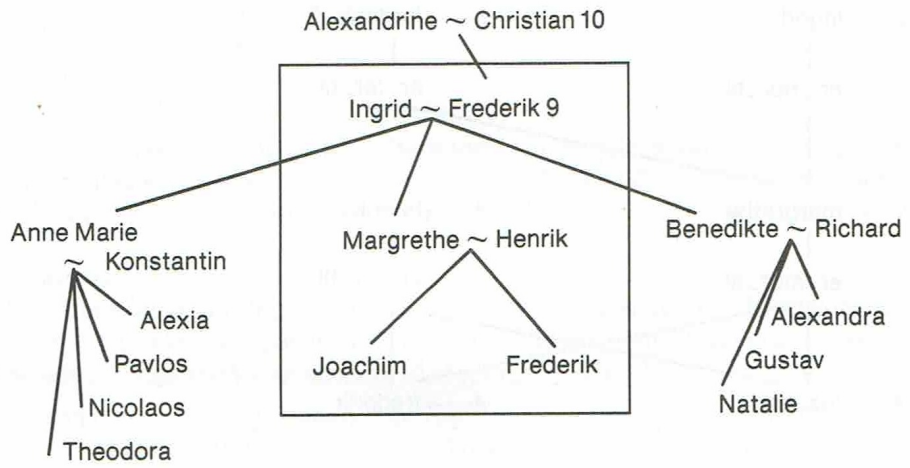
```
mand(henrik).
```

Dette faktum kan læses: Henrik er en mand. Navne på personer skrives med småt, fordi navne med stort begyndelsesbogstav er reserveret til et andet formål, som det vil fremgå senere i dette afsnit. Et faktum som

```
er_mor_til(margrethe,joachim).
```

skal læses som: Margrethe er mor til Joachim.
De familierelationer, vi vil bruge i dette eksempel, er taget fra det ufuldstændige stamtræ i figur 1.1. Fakta for den indrammede del er vist her:

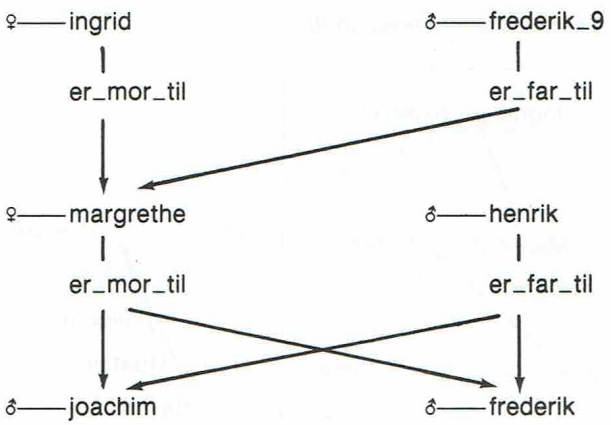
```
mand(henrik).
mand(joachim).
mand(frederik).
```



Figur 1.1 Stamtræ.

```
mand(frederik_9).
kvinde(margrethe).
kvinde(ingrid).
kvinde(anne_marie).
kvinde(benedikte).
er_mor_til(margrethe,frederik).
er_mor_til(margrethe,joachim).
er_mor_til(ingrid,margrethe).
er_mor_til(ingrid,benedikte).
er_mor_til(ingrid,anne_marie).
er_far_til(henrik,frederik).
er_far_til(henrik,joachim).
er_far_til(frederik_9,margrethe).
er_far_til(frederik_9,benedikte).
er_far_til(frederik_9,anne_marie).
```

Vi kan også vise det ved relationer, som vist i figur 1.2.



Figur 1.2 Relationer.

1.2 Spørgsmål til SCANLOG

Når SCANLOG har fået alle oplysningerne om dette stamtræ, er det muligt at spørge om nogle ting. Hvis vi f.eks. vil vide, om Margrethe er mor til Joachim, kan vi stille spørgsmålet:

kan er_mor_til(margrethe,joachim) bekræftes.

hvortil SCANLOG vil svare

Ja

Hvis vi vil have et navn på et af Ingrid's børn, så kan vi spørge SCANLOG

find Barn så er_mor_til(ingrid,Barn).

hvortil SCANLOG vil svare

Et svar er margrethe

For at finde ud af hvem Ingrid er mor til, brugte vi ovenfor en *variabel* ved navn **Barn.** For SCANLOG er navne, der begynder med stort, *variable,* og det er grunden til, at navne på personer blev skrevet med småt. For SCANLOG betyder variablen **Barn,** at der her kan stå hvad som helst, der gør spørgsmålet sandt. I spørgsmålet kom **Barn** til at antage værdien **margrethe. margrethe** er en løsning, for vi har et faktum, der siger

er_mor_til(ingrid,margrethe).

Der kan godt være flere værdier, som gør spørgsmålet sandt (i spørgsmålet ovenfor var **benedikte** og **anne_marie** også løsninger). Hvis vi vil have alle løsninger til ovennævnte spørgsmål, kan vi spørge SCANLOG

find alle Børn så er_mor_til(ingrid,Børn).

hvortil SCANLOG vil svare

Et svar er margrethe
Et svar er benedikte
Et svar er anne_marie
Ikke flere løsninger

Her vil SCANLOG finde ud af, at **Børn** først skal betyde **margrethe,** men da der er flere fakta som passer med **er_mor_til,** så vil SCANLOG finde ud af, at **Børn** også kan være **benedikte** og **anne_marie.**

1.3 Regler

Vi vil finde ud af, om Ingrid er mormor til Joachim. Det kan vi ikke gøre direkte, for vi har ikke fortalt SCANLOG, hvem der er mormor til en person, men vi kan udnytte reglerne for **er_mor_til** til at spørge SCANLOG:

kan er_mor_til(ingrid,margrethe) og
er_mor_til(margrethe,joachim) bekræftes.

hvortil SCANLOG svarer

Ja

I stedet for at skrive alt dette hver gang vi vil stille et spørgsmål, om hvem der er mormor til hvem, så kan vi fortælle SCANLOG, hvordan man kan finde ud af det.

Vi kan formulere det som en regel, at A er mormor til B, hvis A er mor til C, og C er mor til B. Det skriver vi til SCANLOG på denne måde

```
er_mormor_til(A,B) hvis
        er_mor_til(A,C) og
        er_mor_til(C,B).
```

I ovenstående regel svarer A til den, der skal være mormor, og B til barnet, så vi kunne skrive nøjagtigt den samme regel, som vist her

```
er_mormor_til(Mormor,Barn) hvis                    (1)
        er_mor_til(Mormor,Mor) og
        er_mor_til(Mor,Barn).
```

De to regler er helt ens, selv om variabelnavnene er forskellige; i den sidste regel er det dog lettere at se, hvordan variablene er tænkt brugt.

På spørgsmålet

```
find alle Barn så er_mormor_til(ingrid,Barn).
```

vil SCANLOG svare

```
Et svar er frederik
Et svar er joachim
Ikke flere løsninger
```

Når SCANLOG skal finde en løsning til **er_mormor_til(ingrid,Barn),** så skal den finde værdier for de to variable **Mor** og **Barn** i reglen (1), så

```
er_mor_til(ingrid,Mor) og
er_mor_til(Mor,Barn).
```

er sande. SCANLOG finder ud af, at **Mor** kan være **margrethe,** for vi har en regel, der hedder **er_mor_til(ingrid,margrethe).** Alle de steder, hvor **Mor** optræder, indsætter SCANLOG nu **margrethe** og mangler så kun at finde en regel, så

```
er_mor_til(margrethe,Barn).
```

passer. Derfor fremkommer de to løsninger, for både **frederik** og **joachim** kan være en værdi til **Barn.**

> Så snart SCANLOG finder ud af, hvad en variabel kan stå for, bliver værdien indsat alle de steder i spørgsmålet (eller reglen), hvor variabelnavnet optræder.

Der kommer nu andre regler, som også omhandler familieforhold. Bemærk at der i alle reglerne benyttes variable, reglerne gælder derfor for alle familier.
I eksemplerne herunder, er der brugt variabelnavne, der siger, hvad variablene

bruges til. Det gør det lettere at se, hvordan reglen virker, og hvad variablene svarer til. Tegnet »<>«, som forekommer i nogle af reglerne, betyder forskellig fra.

Hvis vi skal afgøre, hvem der er bror til en person, så gør vi det normalt ved:
Hvis A skal være bror til B, så skal A være en mand, og A's mor (C) skal have et barn (B), og B må ikke være den samme som A (man er ikke bror til sig selv).
Det er beskrevet i nedenstående regel; vi har brugt nogle mere sigende variabelnavne end A, B og C.

```
er_bror_til(Bror,Barn) hvis
        mand(Bror),
        er_mor_til(Mor,Bror),
        er_mor_til(Mor,Barn) og
        Bror<>Barn.
er_søster_til(Søster,Barn) hvis
        kvinde(Søster),
        er_mor_til(Mor,Søster),
        er_mor_til(Mor,Barn) og
        Søster<>Barn.
søskende(Barn_1,Barn_2) hvis
        er_bror_til(Barn_1,Barn_2).
søskende(Barn_1,Barn_2) hvis
        er_søster_til(Barn_1,Barn_2).
```

Hvis der er flere regler med det samme navn (som f.eks. **søskende),** så kan det læses, som om der står *eller* mellem dem:

To personer er søskende, hvis den ene er bror til den anden
eller
hvis den ene er søster til den anden.

Vi vil nu lave regler, der finder forfædre. En forfader er enten ens forældre, eller en forfader til ens forældre.
I SCANLOG er det udtrykt sådan:

```
forælder(Far,Barn) hvis er_far_til(Far,Barn).
forælder(Mor,Barn) hvis er_mor_til(Mor,Barn).
forfader(ANE,Person) hvis forælder(ANE,Person).
forfader(ANE,Person) hvis                                    (2)
        forælder(Første_led,Person) og
        forfader(ANE,Første_led).
```

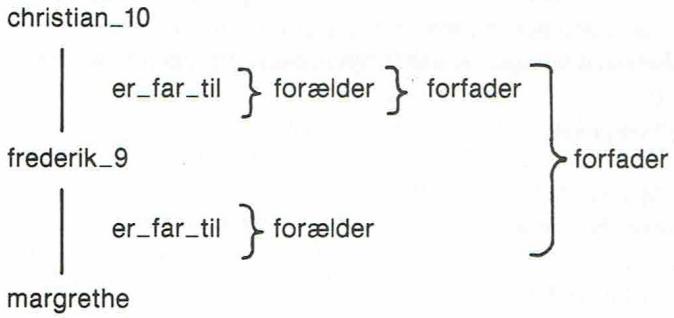
Læg mærke til at vi her løser problemet ved at bruge de samme regler igen, blot på et mindre problem (forældrenes forfædre). Denne metode hedder *rekursion* og vil blive brugt senere gennem bogen.
Vi kan nu stille spørgsmål som

```
kan forfader(christian_10,margrethe) bekræftes.
```

og SCANLOG vil svare

```
Ja
```

Det kan SCANLOG gøre, fordi **frederik_9** er forælder til **margrethe,** og fordi **christian_10** er en forfader til **frederik_9.** Det er også illustreret i figur 1.3.

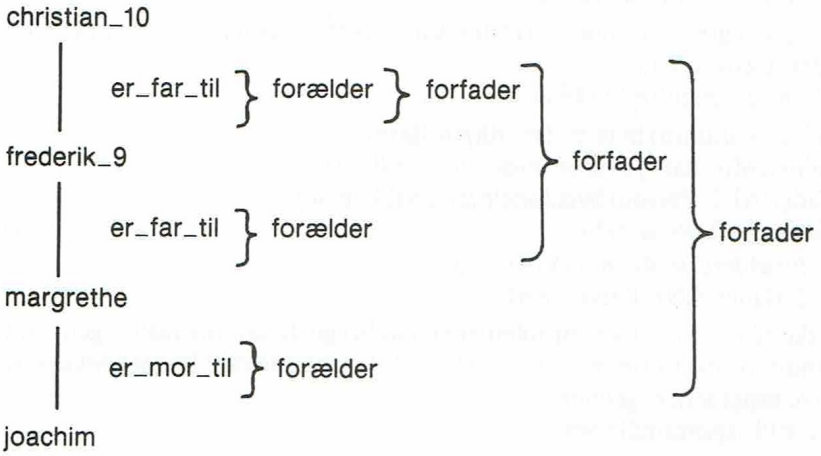


Figur 1.3 **christian_10** forfader til **margrethe.**

Vi kan også spørge SCANLOG:

kan forfader (christian_10,joachim) bekræftes.

Det vil SCANLOG også svare Ja til, for SCANLOG har brugt reglen (2) til at vise, at **margrethe** er forælder til **joachim,** og **christian_10** er en forfader til **margrethe.** Se også figur 1.4.



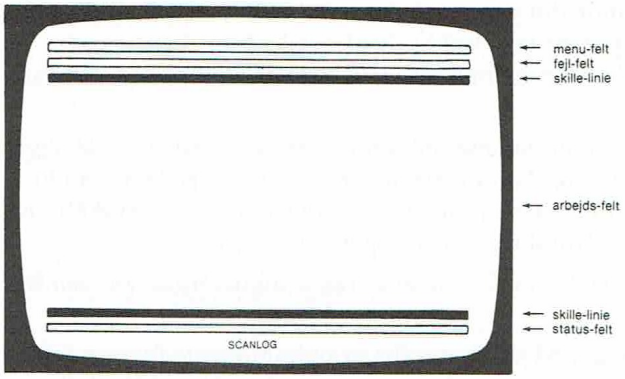
Figur 1.4 **christian_10** forfader til **joachim.**

1.4 Kørselsvejledning

Sæt SCANLOG-disketten i diskettestationen og skriv

SCANLOG

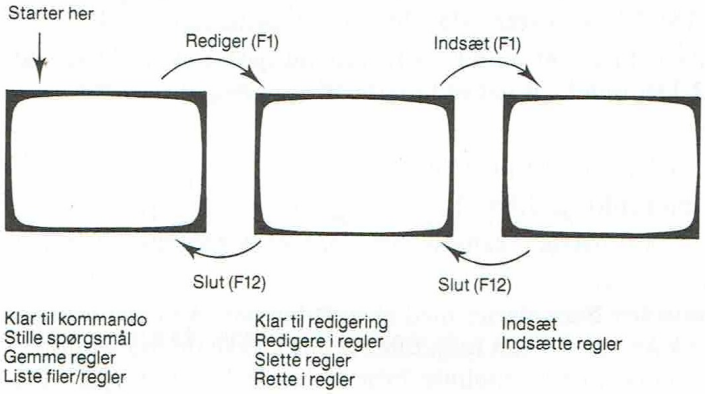
Tryk nu på ←┘-tasten.



Figur 1.5 Skærmbilledet.

Under kørselen med SCANLOG skal du lægge mærke til, hvad der står i skærmbilledets øverste linie *(menulinien)* og næstnederste linie *(statuslinien)*. I *menulinien* kan du se, hvilke kommandoer SCANLOG kan udføre og hvilken tast, der skal trykkes på for at afgive kommandoen. I *statuslinien* kan du se, hvilken tilstand SCANLOG er i. Tilstanden fortæller, hvad SCANLOG laver, eller hvilken kommando der er under udførelse.

I figur 1.6 kan du se nogle af tilstandene, og hvordan man skifter mellem dem. SCANLOG starter i den tilstand, der er markeret med en pil. SCANLOG skifter tilstand, når du afgiver



Figur 1.6 Tilstande i SCANLOG.

nogle bestemte kommandoer, og det er disse, der er skrevet på pilene mellem tilstandene. Under hver tilstand står der, hvad statuslinien indeholder, så du al-

tid kan finde ud af, hvor i systemet du er, og nedenunder står der nogle af de ting, du kan lave i den tilstand.

Efter indtastning af hver regel skal SCANLOG acceptere den. Det vil sige undersøge, om den forstår, hvad du har skrevet. Hvis SCANLOG kan forstå reglen, gemmes den i datamaskinen, så den senere kan bruges.
Under indtastningen kan du bruge markørpilene (←, →, ↑, ↓ og ↖) til at flytte markøren med, men kun indenfor den regel, du er ved at indtaste. Du kan skifte linie ved at trykke på ↵ og fjerne tegn ved hjælp af ← eller **Slet Tegn.**
Du skal ikke være bange for at maskinen går i stykker, når du skriver et eller andet, så kør blot på!

Du kan bede SCANLOG om at indlæse oplysninger fra stamtræet, da de ligger på SCANLOG-disketten. Du får fat i oplysningerne ved at bruge kommandoen **Læs fil (F3).** Tryk på tasten **F3,** og se at menu- og statuslinie skifter. SCANLOG beder om et navn, og her skal du skrive **konge,** og trykke på ↵.

Disse fakta er nu læst ind, og du kan få dem at se ved at afgive kommandoen **Rediger (F1).**

Der fremkommer en del fakta på skærmen; det er oplysningerne fra stamtræet. Du kan komme til at se dem alle ved at bruge ↑ og ↓. Når du har set nok, kommer du videre ved at trykke på **Slut (F12).**

Du kan nu stille SCANLOG nogle af de spørgsmål, som blev beskrevet i det foregående afsnit, for at se om SCANLOG har forstået de indlæste oplysninger. Du kan f.eks. spørge, om Margrethe er mor til Joachim.
Det kan du gøre, ved at trykke på tasten **A1** (i højre side over tallene), derved fremkommer en linie, hvori der står

kan bekræftes.

Fyld linien ud, så der kommer til at stå

kan er_mor_til(margrethe,joachim) bekræftes.

Få SCANLOG til at acceptere spørgsmålet ved at trykke på **Accepter (Tegn Ind).** Efter et stykke tid vil SCANLOG svare »Ja«, hvis du har tastet rigtigt ind.

Hvis du skal finde navnet på et af Ingrid's børn, kan du spørge SCANLOG ved først at trykke på **A2.** Der fremkommer en linie, hvori der står

find så

Fyld denne linie ud, så der kommer til at stå:

find Barn så er_mor_til(ingrid,Barn).

Når du har skrevet variabelnavnet, kan du flytte markøren hen efter **så** ved at trykke på ↓.

Læg mærke til at variablen **Barn** starter med et stort bogstav. Accepter spørgsmålet ved at trykke på **Accepter (Tegn Ind).** Efter et lille stykke tid fremkommer svaret **margrethe,** og du får en ny menulinie, hvor du kan trykke på **Fortsæt (F1)** for at få næste løsning. Svaret **benedikte** vil så fremkomme. Tryk igen på **Fortsæt (F1),** og løsningen **anne_marie** kommer; tryk **Fortsæt (F1),** og SCANLOG vil svare, at der ikke er flere løsninger.

De samme løsninger kan vi få på en anden og lettere måde, ved at spørge SCANLOG efter alle løsninger; tryk på **A3** og fyld denne linie ud

find alle så

så der kommer til at stå

find alle Børn så er_mor_til(ingrid,Børn).

Accepter nu spørgsmålet ved at trykke på **Accepter (Tegn Ind).**
Efter et lille stykke tid, skulle svarene komme, uden at du skal trykke på nogle taster mellem hver løsning.

Du kan indsætte nogle af de regler, der blev lavet i forrige afsnit. For at komme til at indsætte noget, skal du først **Rediger (F1),** og dernæst **Indsæt (F1).** Se evt. figur 1.6.
Du kan indsætte **er_mormor_til**_reglen,

```
er_mormor_til(Mormor,Barn) hvis
        er_mor_til(Mormor,Mor) og
        er_mor_til(Mor,Barn).
```

Accepter reglen med **Accepter (Tegn Ind),** og indtast nedenstående regler (afslut hver enkelt med **Accepter).**

```
er_bror_til(Bror,Barn) hvis
        mand(Bror),
        er_mor_til(Mor,Bror),
        er_mor_til(Mor,Barn) og
        Bror<>Barn.
er_søster_til(Søster,Barn) hvis
        kvinde(Søster),
        er_mor_til(Mor,Søster),
        er_mor_til(Mor,Barn) og
        Søster<>Barn.
søskende(Barn_1,Barn_2) hvis
        er_bror_til(Barn_1,Barn_2).
søskende(Barn_1,Barn_2) hvis
        er_søster_til(Barn_1,Barn_2).
```

Indsæt også reglerne med **forælder** og **forfader** fra side 11. Slut indsætning med **Slut (F12),** og slut redigering med igen at taste **Slut (F12).**

Prøv at stille spørgsmålet **(A3)**

find alle Forfædre så forfader(Forfædre,margrethe).

Du kan også spørge om noget andet, f. eks.

find Efterkommere så forfader(ingrid,Efterkommere).

Der udskrives en løsning, og prøv nu at afgive kommandoen **Hvordan (F6).** SCANLOG udskriver nu, hvordan løsningen er fundet. Du kan bede SCANLOG

finde den næste løsning ved at trykke på **Fortsæt (F1).** Bliv ved at trykke på **Fortsæt (F1)** indtil løsningen **frederik** fremkommer. Prøv at uddybe denne ved at trykke på **Hvordan (F6).** SCANLOG skriver:

```
Det er muligt at bevise
    forfader (ingrid,frederik)
ved at benytte reglen
    forfader(ANE,Person) hvis
        forælder(Første_led,Person) og
        forfader(ANE,Første_led).
og bevise
1:  forælder(margrethe,frederik)
2:  forfader(ingrid,margrethe)
```

Hvis du vil have uddybet, hvorfor 2: gælder, kan du afgive kommandoen **Hvordan (F6)** igen, og SCANLOG spørger, hvilket nummer der skal uddybes, og her skal du skrive 2, og trykke på **Tegn Ind.**
Du kan få SCANLOG til at finde flere løsninger ved at trykke på **Fortsæt (F1),** og hvis du har lyst til at få nogle af svarene uddybet, kan du bruge **Hvordan.** Hvis du ikke vil se flere løsninger, kan du afgive kommandoen **Stop.** Der kommer mere om uddybning i kapitel 2.

Du kan bede SCANLOG om at liste alt, hvad du har fortalt, ved at trykke på **List (F2).** I den øverste linie står der **alt,** og det accepterer du ved at trykke på ←┘.

Du kan gemme alle reglerne på disketten ved at afgive kommandoen **Gem (F4).** SCANLOG foreslår, at alle regler **(alt)** skal gemmes (se øverste linie), og det accepterer du ved at trykke på ←┘. SCANLOG vil nu gerne vide under hvilket navn, den skal gemme reglerne, og som svar herpå kan du f.eks. skrive **kongefam** (afslut med ←┘). Reglerne bliver så gemt på en fil med navnet **kongefam.**

En fil kan betragtes som en kartoteksskuffe, der ligger på disketten, og du kan få SCANLOG til at gemme noget i denne skuffe. Reglerne bliver gemt på en fil, så de kan bruges igen en anden gang, uden at de skal tastes ind forfra. SCANLOG kan indlæse reglerne fra en fil med kommandoen **Læs fil (F3).** Det gjorde du, da oplysningerne om kongefamilien blev læst ind.
Det er muligt at se hvilke filer, der er på disketten ved at afgive kommandoen **List (F2),** skrive **filer,** og trykke på ←┘. SCANLOG spørger om hvilken diskettestation, men tryk blot på ←┘, da tager SCANLOG den diskettestation, du normalt bruger.
Nu skulle det navn **(kongefam),** som du indtastede før gerne være blandt de listede.

Du kan nu stoppe SCANLOG, ved at afgive kommandoen **Slut (F10),** SCANLOG spørger, om du har fået gemt dine regler, og det spørgsmål må du så svare på.

1.5 Resumé

Variable
Variable begynder med et stort bogstav.

Fakta og Regler

Regler (fakta) kan se sådan ud

```
mand(henrik).
er_mor_til(margrethe,joachim).
persondata(ole,klostermarken,4,aalborg).
```

En regel kan også se således ud

```
er_mormor_til(Mormor,Barn) hvis
        er_mor_til(Mormor,Mor) og
        er_mor_til(Mor,Barn).
```

Spørgsmål

Et Ja/Nej spørgsmål er på formen

```
kan er_mormor_til(ingrid,joachim) bekræftes.
```

Et spørgsmål med variable udskrevet er på formen

```
find Bror så er_bror_til(joachim,Bror).
```

Her fås løsningerne een ad gangen.

Et spørgsmål med alle løsninger udskrevet er

```
find alle Far og Barn så er_far_til(Far,Barn).
```

Her kommer alle løsninger på een gang, og der skrives værdier ud for både Far og Barn.

1.6 Opgaver

1. Hvilket spørgsmål skal stilles for at få navnene på alle søskendepar?
2. Hvilket spørgsmål skal stilles for at få navnene på de fædre, som har mere end eet barn?
3. Hvad er forskellen på at stille spørgsmålet?
   1/ find alle Far og Barn så er_far_til(Far,Barn).
   og
   2/ find alle Far så er_far_til(Far,Barn).
   Hvorfor kommer der ingen løsninger til spørgsmålet?
   3/ find alle Far så er_far_til(far,Barn).
4. Brug **Hvordan** til at finde ud af hvordan SCANLOG har fundet en løsning til?

   ```
   find ANE så ANE=alexandrine og
   forfader(ANE,joachim).
   ```

   Hvilke regler er der brugt?
5. Lav regler, der beskriver, hvem der er bedsteforældre.

Opgaverne 9-13 kan kun løses, hvis opgaverne 6-8 er lavet. Man kan i disse opgaver gå sammen holdvis, og derved få flere regler, så opgaverne bliver mere interessante.

6. Vi vil lave nogle regler, der omhandler de enkelte klasser. Vi skal have gemt oplysninger om Siigurd F., som går i 2x, og er naturfaglig. Det kan vi skrive på flg. måde

   klasse(siigurd_F,2,x,mn).

   mn betyder matematisk-naturfaglig
   ms matematisk-samfundsfaglig
   mm matematisk-musisk
   mf matematisk-fysisk
   ns nysproglig
   ss samfundssproglig
   o. s. v.

   Lav regler for jeres klasse, og nogle personer fra andre klasser.
7. Lav nogle regler, som gemmer oplysninger om hvilken lærer, der har en klasse til et fag. F. eks. har 2x fransk ved Lilly Staal.

   lærerplan(2,x,fransk,l_staal).

   Lav regler for din egen og nogle andre klasser.
8. Lav regler for hvilke fag du har, og hvornår du har dem. F. eks. har 2x fransk mandag 10 til 12.

   timeplan(2,x,fransk,mandag,10).
   timeplan(2,x,fransk,mandag,11).

9. Formuler en regel, så man kan finde ud af, om 2 personer går i samme klasse, hvis man kun får deres navne opgivet.
10. Hvilket spørgsmål finder ud af, hvilke fag du har klokken 9?
11. Formuler et spørgsmål, der finder ud af, hvilke lærere du har klokken 9.
12. Prøv at lave nogle regler og et spørgsmål, der som svar giver de fælles lærere, som både du og nogle elever fra en anden klasse har til fælles.
13. Lav et spørgsmål, som kan finde ud af, hvilke andre klasser *din klasse* har timer sammen med.

2. Strenge

Det næste vi skal se på kaldes tekststrenge eller slet og ret strenge.

En streng er en række af tegn omgivet af '. Et eksempel på en streng er

'Hej med dig'

En speciel streng er den tomme streng

''

som ikke indeholder nogen tegn. Bemærk at ' ' ikke er den tomme streng, men strengen bestående af et blankt tegn (mellemrum).

Vi skal i kapitlet se, hvordan SCANLOG stiller strenge og tilhørende primitiver (regler, som kan behandle strenge) til rådighed. Senere i bogen skal vi bl.a. anvende strenge til kodning af tekst og bøjning af franske verber. I dette kapitel skal vi se et eksempel, hvor vi arbejder med små sætninger på engelsk. Sammen med dette eksempel skal vi se to metoder, hvormed vi kan kontrollere den måde, SCANLOG finder løsninger på, og vi skal ikke mindst få SCANLOG til at forklare, **hvordan** en løsning er fundet.

2.1 Et eksempel med strenge

Vores eksempel skal fungere på en sådan måde, at SCANLOG svarer som følger:

```
find alle Svar så svar_på('do you speak french',Svar).
Et svar er 'no I speak german'
Ikke flere løsninger

find alle Svar så svar_på('you are a stupid computer',Svar).
Et svar er 'I am not a stupid computer'
Ikke flere løsninger

find alle Svar så svar_på('do you eat apples',Svar).
Et svar er 'no I eat beans'
Ikke flere løsninger
```

Hvis vi betragter spørgsmålene og svarene nærmere, kan vi begynde at se, hvordan SCANLOG finder svarene. SCANLOG laver simpelthen om på nogle af ordene og genbruger de ord, der ikke genkendes (f.eks. 'speak'). Det kan afsløres ved at stille et spørgsmål som f.eks.

```
find alle Svar så svar_på('I do like  you',Svar).
Et svar er 'I no like  I'
Ikke flere løsninger
```

Bemærk at nogle ord bliver genbrugt, og at blanke også bliver genbrugt. Før vi kaster os ud i en forklaring på reglerne, skal vi se på, **hvordan** et svar er fundet.

Det kan vi få SCANLOG til at fortælle os, og i det næste afsnit er der en kort vejledning til en prøvekørsel med SCANLOG. Man kan eventuelt nøjes med at læse afsnittet, men det vil være en god ide også at prøve det.

2.2 Sådan finder SCANLOG løsningen

På SCANLOG-disketten findes en fil ved navn 'svarpaa', som indeholder reglerne til vores eksempel. Reglerne kan indlæses til SCANLOG ved at afgive kommandoen **Læs fil (F3),** skrive **svarpaa** og trykke på **Tegn Ind.** Efter et øjeblik er reglerne læst ind. For at få et svar og samtidig en forklaring på, hvordan svaret er fundet, afgives et spørgsmål på formen **find så (A2).** Fyld skabelonen ud så der står

```
find Svar så svar_på('do you speak french',Svar).
```

og få SCANLOG til at acceptere spørgsmålet ved at afgive kommandoen **Accepter (Tegn Ind).**

SCANLOG svarer

```
Et svar er 'no I speak german'
```

I menulinien kan du nu se seks forskellige kommandoer. En af kommandoerne hedder **Hvordan (F6),** og den kan give os en forklaring på, hvordan svaret er fundet. Hvis du afgiver kommandoen **Hvordan (F6),** svarer SCANLOG

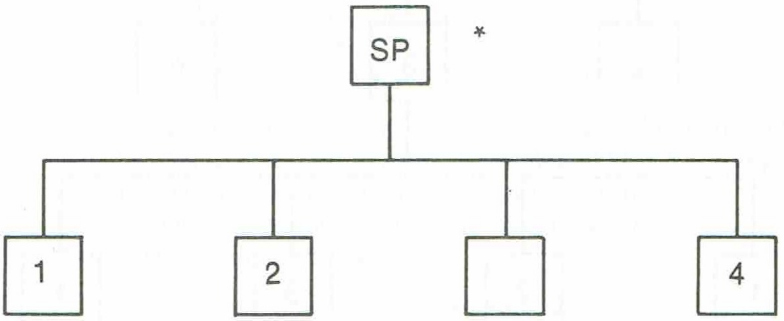
```
Det er muligt at bevise
    svar_på('do you speak french','no I speak german')
ved at benytte reglen
    svar_på(S1,S2) hvis
        første_ord(S1,Ord,Rest),
        lav_om(Ord,Nytord),
        svar_på(Rest,Svar) og
        sammensæt(Nytord,Svar,S2).
```

og bevise

```
1:  første_ord('do you speak french','do', ' you speak french')
2:  lav_om('do','no')
3:  svar_på(' you speak french',' I speak german')
4:  sammensæt('no',' I speak german','no I speak german')
```

Som vi kan se, er der sket en hel del, men vi vil nøjes med at uddybe nummer 3. Nummer 1 tager strengen, som er første argument og finder det første ord i denne samt resten af strengen (det første ord i **'do you speak french'** er **'do'** og resten af strengen er **' you speak french').** Nummer 4 sætter den første og den anden streng sammen (**'no'** sammensat med **' I speak german'** giver **'no I speak german').** Nummer 2 er den del, der laver ordene om i spørgsmålet.

Vi kan illustrere forklaringen på spørgsmålet (vi bruger numrene på de beviste ting), se figur 2.1.



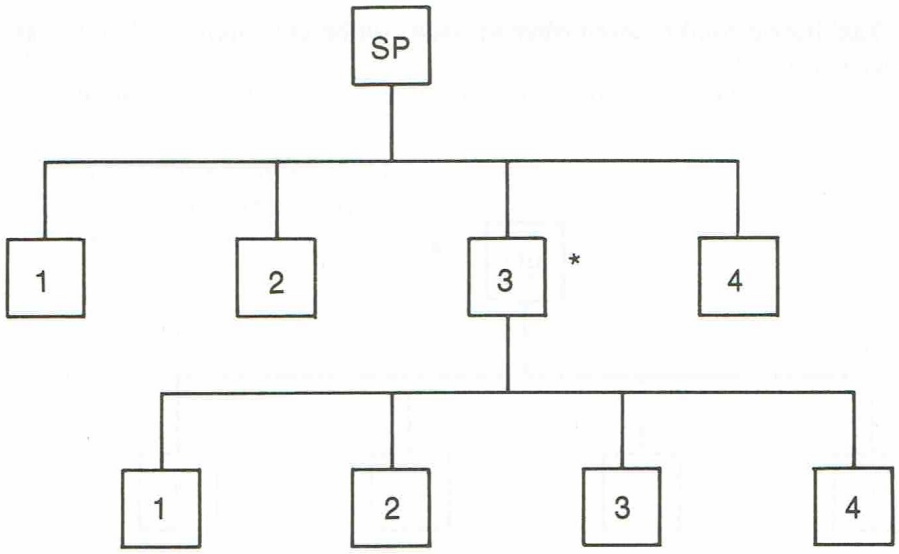
SP: Spørgsmål
* : Uddybes nu

Figur 2.1 Første trin i uddybning.

Du kan nu vælge at få uddybet en af de fire beviste ting, og du kan bede om at få uddybet nummer 3. Det gør du ved at afgive kommandoen **Hvordan (F6),** skrive 3 og trykke på **Tegn Ind.** SCANLOG svarer:

```
Det er muligt at bevise
    svar_på(' you speak french',' I speak german')
ved at benytte reglen
    svar_på(S1,S2) hvis
        første_ord(S1,Ord,Rest),
        lav_om(Ord,Nytord),
        svar_på(Rest,Svar) og
        sammensæt(Nytord,Svar,S2).
og bevise
1.  første_ord(' you speak french',' ','you speak french')
2.  lav_om(' ',' ')
3.  svar_på('you speak french','I speak german')
4.  sammensæt(' ','I speak german',' I speak german')
```

(bemærk at det blanke tegn blev opfattet som et ord, SCANLOG ikke kendte). Hvis vi fortsætter med illustrationen, har vi nu figur 2.2. Som du kan se, kan du fortsætte uddybningen af nummer 3.



SP: Spørgsmål
* : Uddybes nu

Figur 2.2 Andet trin i uddybningen.

Det kan du gøre, indtil SCANLOG svarer

Det er et faktum.

Du kan derefter prøve at bruge kommandoen **Op (F4),** som skridt for skridt bringer dig op igen i modsat retning af den uddybning, du har lavet. Hvad tror du i øvrigt kommandoen **Top (F5)** gør? Når du har set, hvordan reglerne virker, kan du eventuelt undersøge de andre regler. Du kan afslutte uddybningen ved at afgive kommandoen **Fortsæt (F1).** Prøv eventuelt også at uddybe nogle af de andre spørgsmål til SCANLOG.

2.3 En gennemgang af reglerne i eksemplet

Lad os først se på reglerne for **svar_på.** Den første regel er ret simpel. Svaret på den tomme streng er den tomme streng.

```
svar_på(' ',' ').
```

Hvis vi skal finde svaret på en anden streng, som ikke er tom, gøres det ved at lave de enkelte ord om og så sætte svarene sammen i den rigtige rækkefølge.

```
svar_på(S1,S2) hvis
        første_ord(S1,Ord,Rest),
        lav_om(Ord,Nytord),
        svar_på(Rest,Svar) og
        sammensæt(Nytord,Svar,S2).
```

Vi finder altså det første ord, laver det om, finder svaret på resten af teksten og sætter resultatet sammen til et svar. Vi skal nu se nærmere på de enkelte dele.

Lad os først se på reglerne, som laver de enkelte ord om, så SCANLOG svarer som i spørgsmålene side 20. Reglerne skal læses »Lav det første ord om til det andet ord«.

```
lav_om('you','I').
lav_om('are','am not').
lav_om('french','german').
lav_om('do','no').
lav_om('apples','beans').
lav_om(Ukendt,Ukendt).
```

Den sidste regel sørger for, at de ord som SCANLOG ikke kender bliver brugt igen. Det er vigtigt, at denne regel står sidst, for SCANLOG starter med at søge efter en regel, der kan bruges, oppefra og nedefter. Først prøves **lav_om('you','I'),** og hvis ikke den kan bruges, så prøves den næste regel. Den måde, vi har gjort det på, er alligevel ikke helt korrekt, og det kan vi få at se ved at stille spørgsmålet

```
find alle S så lav_om('you',S).
Et svar er 'I'
Et svar er 'you'
Ikke flere løsninger
```

Det er fordi, SCANLOG efter at have fundet og meddelt et svar (Et svar er 'I') søger videre for at se, om der også er en anden løsning. Den sidste regel kan også bruges, og derfor får vi to løsninger. Det er vi ikke interesseret i, og vi må derfor fortælle SCANLOG, at den kun skal vælge een af reglerne for **lav_om.** Det gøres ved at skrive

```
vælg_een lav_om
   lav_om('you','I').
   lav_om('are','am not').
   lav_om('french','german').
   lav_om('do','no').
   lav_om('apples','beans').
   lav_om(Ukendt,Ukendt).
slut.
```

Hvis vi derefter stiller spørgsmålet

```
find alle S så lav_om('you',S).
```

svarer SCANLOG

```
Et svar er 'I'
Ikke flere løsninger
```

vælg_een betyder, at SCANLOG kun må bruge en af de regler, der står mellem **vælg_een** og **slut.** Da SCANLOG prøver reglerne oppefra og nedefter, betyder

det at kun den første brugbare regel bliver brugt. Derfor undgår vi i **lav_om**-reglerne at få to svar, hvoraf det ene ikke ønskes.

Før vi ser på reglerne for **første_ord,** skal vi se nærmere på **sammensæt,** som både kan sammensætte to strenge og finde ud af hvilke strenge, der sammensat giver en tredje streng. **sammensæt** er et primitiv, som SCANLOG stiller til rådighed.

```
kan sammensæt('Hej m','ed dig','Hej med dig') bekræftes.
Ja

find alle S1 og S2 så sammensæt(S1,S2,'Hej med dig').
Et svar er ' ' og 'Hej med dig'
Et svar er 'H' og 'ej med dig'
Et svar er 'He' og 'j med dig'
    .
    .
Et svar er 'Hej med dig' og ' '
Ikke flere løsninger

find alle S så sammensæt(S,'dig','Hej med dig').
Et svar er 'Hej med '
Ikke flere løsninger

find alle S så sammensæt('H',S,'Hej med dig').
Et svar er 'ej med dig'
Ikke flere løsninger
```

første_ord, som finder første ord i en streng, og som leverer resten af strengen med tilbage, kan laves ved hjælp af **find_første** (find det første ord i en streng) og **sammensæt.**

```
første_ord(S1,Ord,Rest) hvis
    find_første(S1,Ord) og
    sammensæt (Ord,Rest,S1).
```

Vi har altså fået regler, som kan pille de enkelte ord ud af en streng. Det kan vi se ved at stille et spørgsmål, hvor vi finder de to første ord i en streng.

```
find alle Ord1 og Ord2 så
    første_ord('Hej med dig',Ord1,Rest1) og
    første_ord(Rest1,Ord2,Rest2).
Et svar er 'Hej' og ' '
Ikke flere løsninger
```

Vi skal også have lavet regler for **find_første,** som finder det første ord i en streng. Først skal vi dog kort se på to andre primitiver, som SCANLOG stiller til rådighed. Det ene er **position.**

```
find alle Pos så position('a','adidas',Pos).
Et svar er 1
Et svar er 5
Ikke flere løsninger
```

position fandt alle forekomster af strengen 'a' i 'adidas'. Den anden vi skal bruge hedder **kopier** (bydemåde af at kopiere).

```
find alle Kopi så kopier ('abekattestreger',4,3,Kopi).
Et svar er 'kat'
Ikke flere løsninger
```

Spørgsmålet ovenfor betød, at vi ville have kopieret fra strengen 'abekattestreger', og kopien skulle starte ved det fjerde tegn og være tre tegn lang.

Vi kan derefter gå i gang med reglerne for **find_første.** Det første ord i en streng, der ikke indeholder blanke, er strengen selv, og den tomme streng indeholder ikke nogen ord.

```
find_første(Ord,Ord) hvis
    Ord<> ' ' og
    ikke(position(' ',Ord,X)).
```

ikke betyder, at der ikke må findes en løsning til det, der står mellem parenteserne.

Det første ord i en streng, der starter med et blankt tegn, er et blankt tegn (vi bruger mellemrum, som om de var ord).

```
find_første(S1,' ') hvis
    sammensæt(' ',X,S1).
```

I de strenge der ikke starter med en blank men indeholder en blank, finder vi det første ord ved at finde positionen af den første blanke og kopiere strengen frem til denne.

```
find_første(S1,Ord) hvis
    ikke(sammensæt(' ',Y,S1)),
    position (' ',S1,X) og
    kopier(S1,1,værdi_af(X-1),Ord).
```

værdi_af betyder, at SCANLOG skal udregne værdien af udtrykket mellem parenteserne. Bemærk hvordan **position** finder værdien af **X,** hvorefter den kan bruges i **kopier.**
Kan du forøvrigt se, hvorfor det ikke er nødvendigt at sige, **S1** <> ' '?

Reglen er desværre ikke helt korrekt. Det er fordi, **position** ikke kun finder den første blanke position i strengen men alle blanke positioner.

```
find alle X så position (' ','Hej med dig',X).
Et svar er 4
Et svar er 8
Ikke flere løsninger
```

Det problem kan vi klare ved at bede SCANLOG om kun at finde een løsning til **position,** og til det formål bruger vi **find_een.**

```
find alle X så find_een(position(' ','Hej med dig',X)).
Et svar er 4
Ikke flere løsninger
```

Den rigtige regel er derfor

```
find_første(S1,Ord) hvis
    ikke(sammensæt(' ',Y,S1)),
    find_een(position(' ',S1,X)) og
    kopier(S1,1,værdi_af(X-1),Ord).
```

Vi har dermed fået lavet reglerne, der finder det første ord i en streng.

2.4 Kørselsvejledning

Reglerne fra kapitlet findes på filen **svarpaa** på SCANLOG-disketten. Prøv at lave en uddybning af spørgsmål med **find_første** og **første_ord.**

2.5 Resumé

Strenge
Strenge er en sekvens af tegn mellem '. SCANLOG stiller primitiver til rådighed, så man kan arbejde med strenge.

sammensæt (S1,S2,S3)
Sandt hvis S1 sammensat med S2 giver strengen S3. Primitivet kan producere løsninger i flere tilfælde.

position (S1,S2,X)
Sand hvis positionen af S1 i S2 er X.

kopier (S1,T1,T2,S2)
Sand hvis kopien af S1 fra tegn nummer T1 og T2 tegn frem er S2.

ikke (Sp)
Sand hvis der ikke findes en løsning til spørgsmålet **Sp. ikke** bruges i regler.

X1 <> X2
Sand hvis X1 ikke er lig med X2, d.v.s. det samme som **ikke (X1=X2).**

find_een (Sp)
Finder een løsning til spørgsmålet **Sp. find_een** bruges i regler.

vælg_een .. slut.
Vælger den første regel mellem **vælg_een** og **slut,** der kan bruges. Resten af reglerne bruges ikke.

Udover disse lavede vi i kapitlet regler **første_ord,** som finder det første ord i en streng, samt resten af strengen. Denne regel kan senere vise sig at være højst anvendelig.

2.6 Opgaver

1. Hvad svarer SCANLOG til spørgsmålet?

   find alle Svar så svar_på('do you like me',Svar).

2. Som du nok har bemærket er vi nødt til f.eks. at skrive 'do you like me' og ikke 'Do you like me'. Tilføj regler til **lav_om,** så SCANLOG også svarer korrekt, når vi skriver ordene med store begyndelsesbogstaver.
3. Der findes endnu to strengprimitiver, som vi ikke har set.
   Kan du se, hvordan de virker?

   find alle S så indsæt('katte','abestreger',4,S).
   Et svar er 'abekattestreger'
   Ikke flere løsninger

   find alle S så slet('abekattestreger',4,5,S).
   Et svar er 'abestreger'
   Ikke flere løsninger

   Du kan eventuelt se side 82 for at kontrollere dine svar. Hvordan virker det tredje strengprimitiv **længde (S1,X)?**
4. Lav regler **højre** og **venstre,** som leverer det sidste henholdsvis det første tegn i en streng. Reglerne skal også levere den streng, der bliver tilbage efter at tegnet er fjernet. På spørgsmålet

   find alle Tegn og Rest så venstre('muldyr',Tegn,Rest).

   skal SCANLOG altså svare

   Et svar er 'm' og 'uldyr'
   Ikke flere løsninger

5. Brug reglerne **højre** og **venstre** i nye regler, **højre_bogstav** og **venstre_bogstav,** som virker på samme måde som **højre** og **venstre,** bortset fra at blanke tegn springes over.
6. Ud fra reglerne **højre_bogstav** og **venstre_bogstav** kan du lave regler, som afgør om en streng er et palindrom.

En streng er et palindrom, hvis det læst forfra og bagfra giver samme mening (hvis vi ser bort fra blanke).

For eksempel er 'en af dem der red med fane' og 'madam adam' palindromer.

7. Prøv om du kan lave tilsvarende regler for **svar_på,** så det foregår på dansk (Det kan være svært).
8. Du kan også bruge strengene til at lave en ord til ord oversættelse af en tekst, f.eks.

    find alle Svar så oversæt('I like you',Svar).
    Et svar er 'Jeg synes om dig'
    Ikke flere løsninger

3. Lister

I dette kapitel skal vi se på lister. En liste er en såkaldt datastruktur, som ofte benyttes i SCANLOG. En liste består af en række af et vilkårligt antal elementer. Netop fordi listen kan indeholde et vilkårligt antal elementer, bliver den meget anvendelig. Desuden udmærker listen sig ved, at elementerne kan være hvad som helst, f.eks. tal eller lister. Strenge, som vi så på i kapitel 2, kunne f.eks. være lavet ved hjælp af lister. I kapitlet skal vi se på, hvad lister er, og vi skal også se nogle af de mest anvendte regler på lister. Reglerne skal alle bruges i sammenhæng med nogle større eksempler senere i bogen, men vi vil ikke se nogen større eksempler i dette kapitel. Ikke desto mindre er reglerne både spændende og interessante og vil senere vise sig højst anvendelige.

3.1 Lister

En liste er betegnelsen for en række elementer omgivet af < og > og adskilt af komma'er, f.eks. er **<abe,kat,mus,gris>** listen bestående af elementerne **abe, kat, mus** og **gris.** Listen uden elementer, den tomme liste, betegnes <>.

Umiddelbart kan man altså ikke se forskel på den tomme liste, <>, og symbolet for forskellig fra, <>. Det vil dog altid fremgå af reglerne, hvad tegnet betyder. Eksemplet nedenunder viser, hvordan man kan binde elementerne i en liste til variable.

```
find alle E1, E2, E3 og E4 så
    <E1,E2,E3,E4>=<abe,kat,mus,gris>.
Et svar er abe, kat, mus og gris
Ikke flere løsninger
```

Der skal være det samme antal variable i den ene liste, som der er elementer i den anden liste.

```
find alle H1 og H2 så <H1,H2>=<1,2,3>.
Ingen løsninger
```

Da en liste kan have et vilkårligt antal elementer, er det nødvendigt at kunne dele lister op i hoved (første element i listen) og hale (resten af listen).

```
find alle Hoved og Hale så
    <Hoved .. Hale>=<abe,kat,mus,gris>.
Et svar er abe og <kat,mus,gris>
Ikke flere løsninger

find alle H1, H2 og Hale så
    <H1,H2 .. Hale>=<abe,kat,mus,gris>.
Et svar er abe, kat og <mus,gris>
Ikke flere løsninger
```

```
find alle H1, H2, H3, H4 og Hale så
    <H1,H2,H3,H4 .. Hale>=<abe,kat,mus,gris>.
Et svar er abe, kat, mus, gris og <>
Ikke flere løsninger
```

Halen til en liste kan godt være den tomme liste, men en liste med hoved og hale kan ikke være tom.

```
find alle Hoved og Hale så <Hoved .. Hale>=<1>.
Et svar er 1 og <>
Ikke flere løsninger

find alle Hoved og Hale så <Hoved .. Hale>=<>.
Ingen løsninger
```

Ovenover så vi, hvordan vi kan få fat på de enkelte elementer i en liste, og hvordan vi kan dele listen op i hoved og hale ved at skrive **<Hoved .. Hale>=<abe,kat,mus,gris>**, hvor **Hoved** bliver første element i listen (hovedet) og **Hale** bliver resten af listen (halen).

Det kan vi udnytte til at finde ud af, om et element er med i en liste.

```
er_med_i(Element,<Element .. Hale>).                    (*)
er_med_i(Element,<Hoved .. Hale>) hvis
        er_med_i(Element,Hale).
```

som siger, at hvis et element er med i en liste, så er det enten hovedet af listen, eller også er det i halen af listen. Bemærk hvordan variablen **Element** i den første regel angiver, at der skal stå det samme på de to steder.
Vi kan derefter spørge, om et element er med i en liste.

```
kan er_med_i(mus,<abe,kat,mus,gris>) bekræftes.
Ja
```

Vi kan også spørge om, hvilke elementer der er i en liste.

```
find alle Element så
    er_med_i(Element,<abe,kat,mus,gris>).
Et svar er abe
Et svar er kat
Et svar er mus
Et svar er gris
Ikke flere løsninger
```

Hvis vi vil vide, hvor mange elementer der er i en liste, kan det gøres ved hjælp af reglerne

```
antal(<>,0).                                            (*)
antal(<Hoved .. Hale>,Antal_Elementer) hvis
    antal(Hale,Antal_i_Halen) og
    Antal_Elementer=værdi_af(Antal_i_Halen+1).
```

som siger, at den tomme liste <> indeholder 0 elementer. Antallet af elementer i en liste, der ikke er tom, findes ved at finde antal elementer i halen og lægge 1 til for elementet i hovedet af listen. Primitivet **værdi_af** *skal* bruges for at udregne værdien af **Antal_i_Halen+1.**

Vi kan for eksempel prøve spørgsmålet

find alle A så antal (<7,9,13>,A).
Et svar er 3
Ikke flere løsninger

Vi kan også sætte lister sammen. Listen <**abe,kat**> sammensat med listen <**mus, gris**> giver listen <**abe,kat,mus,gris**>. Vi kan lave regler, som sætter lister sammen, akkurat som **sammensæt** sætter strenge sammen. Reglerne for at sammensætte (konkatenere) lister er:

konkatener(<>,Liste,Liste). (*)
konkatener(<Element .. H1>,Liste,<Element .. H2>) hvis
konkatener(H1,Liste,H2).

Den første regel siger, at den tomme liste sat foran en anden liste giver listen selv som resultat. Den anden regel sørger for at sætte den første liste forrest i resultatlisten. Det gøres ved at flytte et element ad gangen. For bedre at forstå hvordan det virker, kan vi stille spørgsmålet

find S så konkatener(<abe,kat>,<mus,gris>,S).
Et svar er <abe,kat,mus,gris>

Hvis vi derefter afgiver kommandoen **Hvordan (F6)** svarer SCANLOG

Det er muligt at bevise
konkatener(<abe,kat>,<mus,gris>,<abe,kat,mus,gris>)
ved at benytte reglen
konkatener(<Element .. H1>,Liste,<Element .. H2>) hvis
konkatener(H1,Liste,H2)

og bevise
1: konkatener(<kat>,<mus,gris>,<kat,mus,gris>)

(Bemærk, at variablen **Element** i reglen betyder **abe,** og **H1** betyder <**kat**>).

Vi spørger igen **Hvordan (F6),** og SCANLOG svarer

Det er muligt at bevise
konkatener(<kat>,<mus,gris>,<kat,mus,gris>)
ved at benytte reglen
konkatener(<Element .. H1>,Liste,<Element .. H2>) hvis
konkatener(H1,Liste,H2)

og bevise
1: konkatener(<>,<mus,gris>,<mus,gris>)

Vi prøver igen og får svaret

Det er et faktum.

hvilket jo også passer med vores angivne fakta.

Som vi kan se, sættes listerne sammen ved at flytte elementerne een ad gangen fra den første liste over i resultatlisten, og derefter afslutte sammensætningen ved at gøre den anden liste til resultatlistens hale.

konkatener kan bruges til at finde, hvilken liste der sammensat med en anden giver en tredje liste.

find alle X så
konkatener(X,<gris>,<abe,kat,mus,gris>).
Et svar er <abe,kat,mus>
Ikke flere løsninger

find alle X så
konkatener(<abe,kat>,X,<abe,kat,mus,gris>).
Et svar er <mus,gris>
Ikke flere løsninger

konkatener kan også bruges til at finde alle lister, som sammensat giver en anden liste, på samme måde som **sammensæt** kan på strenge.

find alle L1 og L2 så konkatener(L1,L2,<7,9,13>).
Et svar er <> og <7,9,13>
Et svar er <7> og <9,13>
Et svar er <7,9> og <13>
Et svar er <7,9,13> og <>
Ikke flere løsninger

3.2 Kørselsvejledning

I kapitlet er alle regler, der skal bruges, mærket med en stjerne(*). Det vil være en god ide at taste dem ind og få dem gemt på en fil på disketten. Især **er_med_i** og **konkatener** som skal bruges senere. Prøv også at stille spørgsmål fra kapitlet og få uddybet nogle af svarene. Når du får uddybet et svar og er nået til det punkt, hvor SCANLOG svarer

Det er et faktum.

kan du få SCANLOG til at fortsætte med at søge efter løsninger ved at afgive kommandoen **Fortsæt (F1).**

3.3 Resumé

Lister
Lister er en række af elementer indeholdt mellem < og >. **<abe,kat,mus, gris>** er listen bestående af elementer **abe, kat, mus** og **gris.** En liste kan indeholde et vilkårligt antal elementer. En speciel liste er den tomme liste, <>.

Hoved . . Hale
En liste kan deles op i hoved (det første element af listen) og hale (resten af listen). For eksempel vil **<Hoved .. Hale>=<1,2,3>** give resultatet **Hoved=1** og **Hale=<2,3>.**

De vigtigste regler, vi har set i dette kapitel, er **er_med_i** og **konkatener,** som vi senere skal anvende i nogle større eksempler. I kapitlet har vi set lister med elementer på formen **abe, kat** o. s. v., men elementerne kan også være tal, strenge eller for den sags skyld lister. I opgaverne nedenunder kan du finde forskellige måder at bruge lister på.

3.4 Opgaver

1. I stedet for at skrive

   er_mor_til(margrethe,frederik).
   er_mor_til(margrethe,joachim).

   hvor man angiver Margrethe to gange, kan man bruge lister til at angive et **er_mor_til**-faktum (som måske kunne hedde **har_børn).** Hvordan kan det gøres?
2. Hvordan kan man lave en **er_mor_til**-regel ud fra **har_børn** fra forrige opgave?
3. Lav en regel, som finder halen af en liste, det vil sige, så SCANLOG til spørgsmålet

   find alle Hale så find_hale(<byg,rug,hvede>,Hale).

   svarer

   Et svar er <rug,hvede>
   Ikke flere løsninger
4. Hvordan kan man lave en regel, som sætter et element forrest i en liste, så SCANLOG til spørgsmålet?

   find alle S så tilføj_een(3,<7,9,13>,S).

   svarer

   Et svar er <3,7,9,13>
5. I kapitlet så du reglerne for **er_med_i,** som finder alle elementer, der er med i en liste. Prøv om du kan lave regler, som finder et element på en bestemt plads i en liste, d. v. s. så SCANLOG til spørgsmålet?

   find alle E så plads(3,<abe,mus,kat,gris>,E)

   svarer

   Et svar er kat
   Ikke flere løsninger

   som er på den tredje plads i listen.

6. På næsten samme måde som vi lavede **er_med_i,** kan du lave regler som finder ud af, om to elementer står ved siden af hinanden i en liste (er naboer).
7. Lav regler som vender en liste om *(reverserer* den), f.eks. laver listen <7,9,13> om til listen <13,9,7,>. Til det formål kan du bl.a. bruge **konkatener-reglerne.** (Vink: Hvis man vil vende listen <7,9,13> om, kan det gøres ved at vende halen <9,13> om og sætte listen indeholdende hovedet 7 efter resultatet <13,9>).
8. Lav regler som finder det sidste element i en liste, d.v.s. så SCANLOG til spørgsmålet

   find alle E så sidste(E,<talk,of,the,town>).

   svarer

   Et svar er town
   Ikke flere løsninger

9. Prøv om du kan lave regler, som udskifter *(substituerer)* et bestemt element i en liste med et andet element, d.v.s. så SCANLOG til spørgsmålet

   find alle S så
   substituer(vand,øl,<jeg,kunne,drikke,10,vand>,S).

   svarer

   Et svar er <jeg,kunne,drikke,10,øl>
   Ikke flere løsninger

   d.v.s. SCANLOG har udskiftet alle forekomster af vand med øl i listen.
10. Brug **konkatener** til at lave regler for **er_med_i, sidste** og **naboer.**
11. En liste kan repræsentere en mængde, f.eks. kan listen <1,2,3> repræsentere mængden {1,2,3}. Betragtes 2 mængder *M1* og *M2* siges *M1* at være en delmængde af *M2,* hvis alle elementer i *M1* er med i *M2.*. Den tomme mængde er med i enhver anden mængde. Lav reglerne for delmængde, så SCANLOG svarer Ja, hvis den første liste (mængde) er indeholdt i den anden liste (mængde).
12. Prøv, som i opgave 11, at lave andre regler om mængder, f.eks. fællesmængde, mængdedifferens og måske krydsprodukt mellem to mængder. Et krydsprodukt mellem to mængder *M1* og *M2* er mængden bestående af alle par *(m1,m2),* hvor *m1* tilhører *M1* og *m2* tilhører *M2.* Krydsproduktet mellem {7,9,13} og {1,2} er således {(7,1), (7,2), (9,1), (9,2), (13,1), (13,2)}. Reglerne skal laves så SCANLOG til spørgsmålet

    find alle S så krydsprodukt(<7,9,13>,<1,2>,S).

    svarer

    Et svar er <<7,1>,<7,2>,<9,1>,<9,2>,<13,1>,<13,2>>
    Ikke flere løsninger

4. SCANLOG's løsningsmetode

Når vi laver regler, kan det ofte være nødvendigt at kende den løsningsmetode, som SCANLOG anvender. Der er tre ting, som vi skal kende til:

1. den rækkefølge, hvori reglerne anvendes.
2. hvordan reglerne spiller sammen.
3. hvordan SCANLOG finder flere forskellige løsninger.

Vi skal også se, hvordan variable anvendes, og hvad de betyder. Sidst i kapitlet skal vi desuden se to eksempler på, hvordan og hvorfor det kan gå galt.

4.1 Hvad hedder det?

Som vi har set findes der to former for regler,

```
er_mor_til(ingrid,benedikte).
```

og

```
er_farfar_til(Farfar,Barn) hvis
    er_far_til(Farfar,Far) og
    er_far_til(Far,Barn).
```

Den første regel kaldes også for et faktum. Hvis vi ser på den anden regel, så kaldes det der står før **hvis** for reglens *hoved* (konklusion), og det efter **hvis** kaldes for reglens *krop* (betingelserne). Den anden regel kan også skrives på formen

```
hvis er_far_til(Farfar,Far) og
    er_far_til(Far,Barn) så
    er_farfar_til(Farfar,Barn).
```

som betyder nøjagtigt det samme som den anden **er_farfar_til**-regel. Man har derfor mulighed for at skrive sine regler alt efter, hvordan en løsning bedst formuleres. Nogle problemer lægger op til, at opfyldte betingelser giver en konklusion **(hvis så),** mens andre lægger op til, at noget gælder, hvis bestemte betingelser er opfyldt. SCANLOG tillader begge skrivemåder og giver dermed mulighed for at lave regler, som er lette at læse og forstå.

er_mor_til og **er_farfar_til** kaldes for en *relation* eller et *prædikat.* En relation gælder mellem de argumenter (parametre) relationen har. Således er **ingrid** første parameter og **benedikte** anden parameter i relationen **er_mor_til** ovenover. De enkelte relationer i en regels krop kaldes også for delproblemer.

4.2 Udvælgelse af regler

Når SCANLOG skal finde en løsning udvælges reglerne i den rækkefølge, de er indsat i, ovenfra og ned. Hvis vi har reglerne

```
kvinde(mie).
kvinde(pia).
kvinde(camilla).
```

og stiller spørgsmålet

```
find alle Navn så kvinde(Navn).
```

får vi svarene i den samme rækkefølge som ovenfor

```
Et svar er mie
Et svar er pia
Et svar er camilla
Ikke flere løsninger
```

I dette tilfælde er rækkefølgen ikke så vigtig, men vi har allerede set et tilfælde, hvor den betød meget. Det var reglerne for **lav_om**

```
vælg_een lav_om
   lav_om('you','I').
   lav_om('are','am not').
   lav_om('french','german').
   lav_om('do','no').
   lav_om('apples','beans').
   lav_om(Ukendt,Ukendt).
slut.
```

hvor reglen **lav_om(Ukendt,Ukendt)** skal stå sidst. Hvis vi havde anbragt denne som den første regel, ville alle ord blive genbrugt og ingen ville blive lavet om (fordi de var i en **vælg_een).** En tommelfingerregel er at indsætte alle specialtilfælde før de generelle regler, nøjagtigt som vi gjorde med **lav_om**-reglerne. Der så vi også, hvordan vi kunne kontrollere udvælgelsen af regler ved hjælp af **vælg_een.** Når SCANLOG prøver regler i en **vælg_een,** så vælges kun den første, hvor hovedet (konklusionen) ser ud til at kunne anvendes.

4.2.1 Sådan virker vælg_een

Det er ikke altid tilstrækkeligt, at konklusionen ser ud til at kunne anvendes. Til tider kan det være praktisk at knytte betingelser på konklusionen. Derfor kan enhver regel have en **hvor**-del. Et eksempel kan være to regler med hoved, **beregn (Tal1,Tal2,Resultat),** hvor den første skal anvendes, hvis **Tal1** er større end **Tal2,** og ellers skal den anden regel anvendes. Reglerne kan skrives som

```
beregn(Tal1,Tal2,Resultat) hvis
     Tal1 > Tal2 og
     ...
beregn(Tal1,Tal2,Resultat) hvis
     Tal1 < = Tal2 og
     ...
```

I **vælg_een** vælger SCANLOG den første regel, hvor konklusionen kan bruges *og* **hvor**-delen kan bevises. Reglerne kan så skrives som

```
vælg_een beregn
   beregn(Tal1,Tal2,Resultat) hvor
   Tal1 > Tal2, hvis
   ...
   beregn(Tal1,Tal2,Resultat) hvis
   ...
slut.
```

Bemærk at det ikke er nødvendigt at specificere **Tal1 <= Tal2** i den anden regel i **vælg_een. hvor**-delen vil blive anvendt i senere eksempler, hvor det vil blive mere klart, hvorfor og hvordan man bruger den. Se eventuelt også afsnit 7.3.8 »Lidt om hvor-delen« i opslagsværket sidst i bogen.

4.3 Bevisførelse

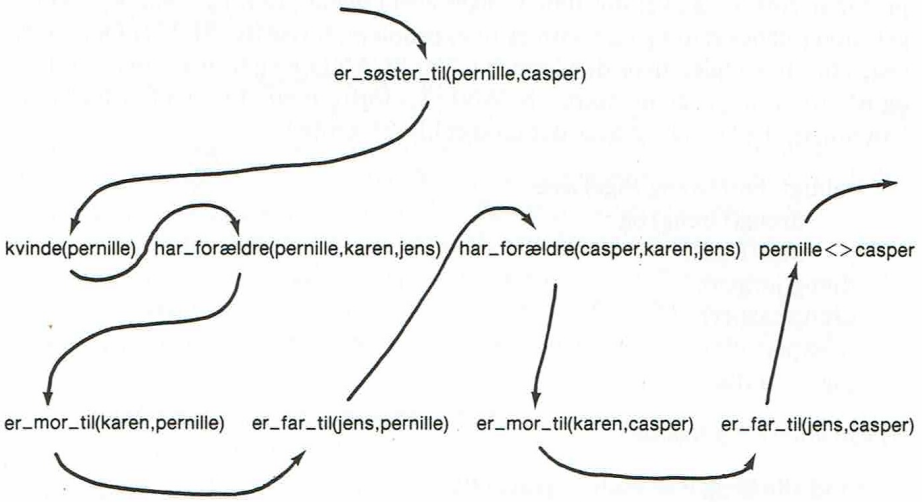
Når SCANLOG skal bevise en regel anvendes en strategi kaldet *dybde-først.* Strategien går ud på, at SCANLOG beviser en regel til bunds, før en ny regel prøves. Man siger også, at en regel bevises fra venstre mod højre. Hvis vi betragter reglen

```
er_søster_til(Søster,Søskend) hvis
    kvinde(Søster),
    har_forældre(Søster,Mor,Far),
    har_forældre(Søskend,Mor,Far) og
    Søster <> Søskend.
```

og har stillet spørgsmålet

```
kan er_søster_til(pernille,casper) bekræftes.
```

så vil SCANLOG bevise det på følgende måde.



Figur 4.1 Dybde-først.

Først prøver SCANLOG at bevise **kvinde(pernille).** Hvis det går godt forsøger SCANLOG at bevise **har_forældre(pernille,Mor,Far).** Hvis også dette går godt (lad os sige, at mor er **Karen,** og far er **Jens**) fortsættes med **har_forældre(casper, karen,jens).** Til sidst kommer SCANLOG til **pernille <> casper,** hvilket er sandt, og SCANLOG svarer Ja.

Vi kan betragte reglerne nedenunder, som er nok til at løse spørgsmålet.

```
kvinde(pernille).
er_mor_til(karen,pernille).
er_mor_til(karen,casper).
er_far_til(jens,pernille).
er_far_til(jens,casper).
har_forældre(B,M,F) hvis
    er_mor_til(M,B) og
    er_far_til(F,B).
```

Hvis vi følger pilen på figur 4.1, kan vi se, i hvilken rækkefølge SCANLOG har løst de enkelte problemer.

4.4 Flere løsninger

Når SCANLOG skal finde flere løsninger, gøres det ved en strategi kaldet *tilbagesporing.* Tilbagesporing går ud på, at SCANLOG på et tidspunkt vender tilbage til et problem, der er løst én gang, og SCANLOG prøver så at finde en anden løsning. Hvis vi kigger på figur 4.1 betyder det, at SCANLOG følger linien tilbage imod pilenes retning og forsøger at finde en ny løsning, hver gang der kommer et delproblem. Hvis en sådan findes, fortsætter SCANLOG med næste problem, som vi så ovenfor (linien følges igen i pilenes retning). Når SCANLOG går mod pilenes retning og kommer til et problem, fortsætter SCANLOG med at søge efter nye regler, hvor den kom fra. Når SCANLOG går med pilenes retning og når til et delproblem, starter SCANLOG forfra med at søge efter regler, der kan bruges. Det er lettest at se det med et lille eksempel.

```
muligt_par(Dreng,Pige) hvis
    dreng(Dreng) og
    pige(Pige).
dreng(jørgen).
dreng(casper).
pige(pernille).
pige(camilla).
```

Vi kan stille et spørgsmål.

```
find alle D og P så muligt_par(D,P).
Et svar er jørgen og pernille
Et svar er jørgen og camilla
Et svar er casper og pernille
Et svar er casper og camilla
Ikke flere løsninger
```

Bemærk rækkefølgen af svarene. SCANLOG finder svarene ved at anvende reglen for **muligt_par.** Først findes en dreng (det bliver **jørgen**), og SCANLOG fortsætter med det næste problem, at finde en pige. Den første pige er **pernille,** og

dermed er det første svar fundet. Nu benytter SCANLOG tilbagesporing. SCANLOG 'glemmer' den sidste fundne løsning på **Pige** og søger efter andre løsninger. Den næste løsning er **camilla,** og det andet svar er fundet. SCANLOG benytter igen tilbagesporing og forsøger at finde en ny løsning til **Pige,** men der er ikke flere. Derfor benyttes endnu en gang tilbagesporing, og SCANLOG kommer tilbage til problemet med at finde **Dreng.** Den sidst fundne løsning var **jørgen,** og SCANLOG søger videre for at se, om der er flere løsninger. **casper** er en anden løsning, og SCANLOG fortsætter nu med det næste problem, at finde en pige. Nu går det som før, først findes **pernille,** svaret rapporteres, SCANLOG benytter tilbagesporing, finder **camilla** og svaret rapporteres. Derefter benytter SCANLOG igen tilbagesporing, men nu er der ikke flere piger. SCANLOG benytter tilbagesporing endnu en gang og kommer tilbage til dreng. Men der er heller ikke flere drenge, og SCANLOG rapporterer, at der ikke er flere løsninger. Under løsning af spørgsmål kan du se, hvordan SCANLOG forsøger at finde en løsning. Tegnet i meddelelsesfeltet løber fra venstre mod højre, når SCANLOG er i gang med dybde-først, og det løber den modsatte vej, når SCANLOG benytter tilbagesporing.

4.5 Variable

Vi skal også vide, hvordan SCANLOG bruger variable.

> En variabel kan enten være fri eller bundet. En variabel er fri, hvis den endnu ikke har fået en værdi. En variabel er bundet, hvis SCANLOG har fundet noget, den skal betyde.

Når vi stiller spørgsmålet

```
find alle Dame så kvinde(Dame).
```

er variablen **Dame** *fri,* men når SCANLOG finder det første **kvinde**-faktum (f. eks. **kvinde(margrethe)**), bliver variablen *bundet* og kommer til at betyde **margrethe.**

En variabel fungerer som en *pladsholder* indenfor en enkelt regel. Hvis vi ser på reglen

```
er_søster_til(Søster,Barn) hvis
    kvinde(Søster),
    har_forældre(Søster,Mor,Far),
    har_forældre(Barn,Mor,Far) og
    Søster <> Barn.
```

så betyder variablen **Søster,** at der på alle pladser, hvor variabelnavnet optræder, skal stå det samme. Hvis vi stiller spørgsmålet

```
kan er_søster_til(pernille,casper) bekræftes.
```

kommer **Søster** til at betyde **pernille,** og **Barn** betyder **casper.**

Dermed får vi faktisk en anvendelse af reglen på formen

```
er_søster_til(pernille,casper) hvis
    kvinde(pernille),
    har_forældre(pernille,Mor,Far),
    har_forældre(casper,Mor,Far) og
    pernille <> casper.
```

Da en variabel fungerer som pladsholder indenfor en enkelt regel, så kan den naturligvis kun betyde noget indenfor en enkelt regel. I de to regler

```
søskende(Bror,Barn) hvis
    er_bror_til(Bror,Barn).
søskende(Søster,Barn) hvis
    er_søster_til(Søster,Barn).
```

har variablen **Barn** i den første regel *intet* at gøre med variablen **Barn** i den anden regel.
Hvad sker der så, når SCANLOG anvender den samme regel flere gange under et bevis? Når SCANLOG skal bruge en regel, så laver den simpelthen en kopi af reglen. Derfor har SCANLOG hver gang en ny kopi af reglen med nye variable, og reglerne kan anvendes flere gange under et bevis.

4.6 Løkker

En hyppig fejl i regler er, at SCANLOG aldrig finder en løsning men bliver ved med at lede efter den i en uendelighed. Det kaldes en løkke, og i dette afsnit skal vi se et par eksempler på løkker, og hvordan de undgås.

4.6.1 Symmetriske relationer

En ting vi skal passe på er såkaldte symmetriske relationer. Et eksempel på en symmetrisk relation er

```
er_gift_med(margrethe,henrik).
```

som siger, at Margrethe er gift med Henrik. Relationen er i daglig sprog symmetrisk, for Henrik er jo også gift med Margrethe. Det er derfor uheldigt, at vi til spørgsmålet

```
kan er_gift_med(margrethe,henrik) bekræftes.
```

får svaret Ja, mens vi til spørgsmålet

```
kan er_gift_med(henrik,margrethe) bekræftes.
```

får svaret Nej.

Der er to måder at tackle problemet på. Den ene er at angive både at Margrethe er gift med Henrik, og at Henrik er gift med Margrethe. Denne fremgangsmåde er arbejdskrævende, hvis vi har mange par. Den anden måde er at lave en regel, som bytter om på parametrene.

```
er_gift_med(A,B) hvis
    er_gift_med(B,A).
```

Uheldigvis betyder denne regel, at SCANLOG kan blive ved med at bytte om på parametrene i en uendelighed (hvis SCANLOG ikke ved, at et par er gift, vil den altid anvende denne regel), så der skal mere til.
I vores **er_gift_med**-relation er første parameter en kvinde. Reglen, som bytter om på parametrene, skal altså kun bruges, hvis anden parameter er en kvinde. Da vi i forvejen har regler for **kvinde,** kan vi lave reglen.

```
er_gift_med(Mand,Kvinde) hvis
    kvinde(Kvinde) og
    er_gift_med(Kvinde,Mand).
```

Hvad sker der så, når vi stiller spørgsmålet

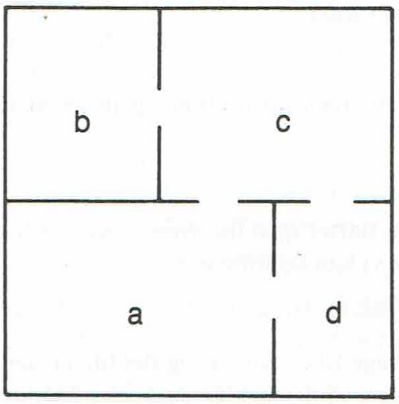
```
kan er_gift_med(gurli,maren) bekræftes.
```

SCANLOG kan igen blive ved med at bytte om på parametrene i en uendelighed (forudsat at der er fakta **kvinde(maren)** og **kvinde(gurli)**). Vi er også nødt til at sikre os, at den anden parameter er en mand. Den korrekte regel er

```
er_gift_med(Mand,Kvinde) hvis
    kvinde(Kvinde),
    mand(Mand) og
    er_gift_med(Kvinde,Mand).
```

4.6.2 »Fra start til slut spil«

Lad os betragte et lille eksempel, hvor SCANLOG's løsningsmetode kan betyde, at det går galt, hvis vi ikke passer på. Vi betragter en lille labyrint (figur 4.2), hvor vi står i rummet a og skal hen til rummet b.



Figur 4.2 Labyrint.

Vi kunne så fortælle SCANLOG mellem hvilke rum, der er en dør (der findes et træk fra et rum til et andet).

```
træk(a,c).
træk(a,d).
træk(b,c).
træk(c,a).
træk(c,d).
træk(c,b).
træk(d,c).
træk(d,a).
```

Bemærk at relationen **træk** er symmetrisk, men at vi har angivet alle muligheder, altså 8 regler for 4 døre. Derefter kunne vi lave regler, som finder en sekvens af rum, så vi kommer fra a til b. Vi kunne gemme rummene i en liste, hvor det første element betegner det rum, vi står i. Metoden til at finde en løsning kunne være, at vi går ind i et rum og prøver at finde frem til rum b fra dette rum. Hver gang vi kommer ind i et nyt rum, skal vi vælge mellem en eller flere muligheder for det næste rum. Det gør vi ved at lade SCANLOG vælge for os i **træk**-reglerne. Hvis det viser sig, at vi ikke kan komme til rum b, går vi tilbage og beder SCANLOG om at vælge et nyt rum. SCANLOG vælger så den næste **træk**-regel, som kan bruges. Nedenunder bruger vi denne metode på labyrinten.

Fra start vil vi have

```
<a>
```

altså vi starter i a. Vi kan derefter gå til c, altså

```
<c,a>
```

og til sidst havne i b

```
<b,c,a>
```

Lad os se, hvad SCANLOG ville gøre. Vi starter med

```
<a>
```

og skal lave et træk. SCANLOG starter for oven i reglerne fra træk og finder, at vi kan gå fra a til c.

```
<c,a>
```

SCANLOG skal derefter finde et træk fra c og starter igen fra oven i reglerne for træk. Den første regel, der kan bruges, siger, at vi kan komme til a.

```
<a,c,a>
```

Som vi kan se vil SCANLOG derefter gå tilbage til c, så til a, og det bliver den ved med og finder ikke nogen løsning. Vi siger, at der opstår en *løkke*. Måden hvorpå dette undgås er ganske simpel. Vi skal blot tilføje, at SCANLOG ikke må gå tilbage til et rum, hvor den allerede har været, eller med andre ord, når SCAN-

LOG finder et nyt rum, så må det ikke være med i listen af rum, vi har besøgt. Lad os prøve det.

Vi starter med

<a>

Derefter kan vi gå til c, for c er ikke med i listen af rum. Vi har så

<c,a>

Fra c vil SCANLOG gå tilbage til a, men a er med i listen, så SCANLOG benytter tilbagesporing og finder næste mulighed, d.

<d,c,a>

Fra d kan SCANLOG komme til a eller c, men begge har været med før. Altså var det forkert at gå fra c til d, og SCANLOG benytter tilbagesporing og vender tilbage til

<c,a>

SCANLOG har allerede prøvet at gå fra c til a og fra c til d. Derfor prøver SCANLOG næste mulighed, nemlig fra c til b.

<b,c,a>

SCANLOG har dermed fundet en løsning. Nedenunder skal vi se de regler, som kan finde frem til svaret.

4.6.3 Regler for »fra start til slut spil«

I sidste afsnit så vi, hvordan man kan finde en løsning til labyrinten. I dette afsnit vil vi lave regler for løsningsmetoden, så SCANLOG finder løsningen til os. Vi skal lave regler **udvid(Mellem_Res,Resultat,Slutrum),** som finder en liste af rum, så vi f.eks. kan komme fra a til b. Første parameter betegner en liste af de rum, vi hidtil har besøgt i rækkefølge, så hovedet er det rum, vi nu er i; anden parameter skal være resultatet, og tredje parameter fortæller, hvor vi skal slutte (Til labyrinten starter vi med **udvid(<a>,Resultat,b)**). Den første regel for **udvid** beskriver, hvornår vi har fundet en løsning, nemlig når hovedet af listen er slutrummet.

```
udvid(Løsning,Løsning,Slutrum) hvis
    starter_med(Løsning,Slutrum).
```

hvor **starter_med** betyder, at listen som er første parameter starter med elementet, som er anden parameter. Reglen, som finder ud af, hvad en liste starter med, er

```
starter_med(<Hoved .. Hale>,Hoved).
```

Den anden regel for **udvid** tilføjer nye rum.

```
udvid(L1,Løsning,Slutrum) hvis
    udvid_med_et_træk(L1,Udvidet),
    løkke_fri(Udvidet) og
    udvid(Udvidet,Løsning,Slutrum).
```

Vi udvider med et træk ved at finde ud af, hvor vi er og finde et træk derfra.

```
udvid_med_et_træk(Gl_liste,<Nyt_rum .. Gl_liste>) hvis
    starter_med(Gl_liste,Sidste_rum) og
    træk(Sidste_rum,Nyt_rum).
```

Reglen, som sikrer at der ikke er løkker, kan laves med **er_med_i.**

```
løkke_fri(<Hoved .. Hale>) hvis
    ikke(er_med_i(Hoved,Hale)).
```

Vi kan nu stille spørgsmålet

```
find alle Svar så udvid(<a>,Svar,b).
Et svar er <b,c,a>
Et svar er <b,c,d,a>
Ikke flere løsninger
```

Reglerne for **udvid** ligger på filen **udvid** på SCANLOG-disketten.

4.7 Opgaver

1. Hvorfor er der ikke en **hvor**-del på den anden regel i **vælg_een**-reglerne for **beregn?**
2. Hvad ville der ske, hvis vi i stedet for **vælg_een**-reglerne på side 36 skrev?

```
vælg_een beregn
    beregn(Tal1,Tal2,Resultat) hvis
    Tal1 > Tal2 og
    ...
    beregn(Tal1,Tal2,Resultat) hvis
    ...
slut.
```

3. Et andet fra start til slut spil er flodpassage-problemet. En bonde, en ulv, en ged og et kålhoved står på nordbredden af en flod og skal til sydbredden. Ved floden ligger en robåd, som bonden kan ro over i. Han kan højst have een passager med sig i båden (kålhovedet er også en passager). Hvordan kan de komme over, når følgende skal tages i betragtning: hvis bonden er på den ene bred, og ulven og geden på den anden bred, æder ulven geden, og hvis bonden er på den ene bred, og geden og kålhovedet er på den anden bred, æder geden kålhovedet.
   For at løse dette problem skal vi have en måde at repræsentere deres position på, nøjagtigt som med rummene i labyrinten. Lad derfor

   ```
   p(Bonde,Ulv,Ged,Kål).
   ```

   betyde

   bonden er på **Bonde**-bredden,
   ulven er på **Ulv**-bredden,
   geden er på **Ged**-bredden og
   kålhovedet er på **Kål**-bredden,

hvor **Bonde, Ulv, Ged** og **Kål** betyder nord(**n**) eller syd(**s**). Således betyder **p(n,s,n,s)** at bonden er på nord-bredden, ulven er på sydbredden o. s. v.

Fra start er de alle på nordbredden,

p(n,n,n,n)

og skal slutte på sydbredden

p(s,s,s,s)

Vi skal derefter beskrive de lovlige træk, der kan udføres. Det nemmeste er at beskrive de ulovlige og ud fra disse finde de lovlige. Der er 2 ulovlige tilstande, og reglen for den ene er

ulovlig(Bonde,Ulv_ged,Ulv_ged,Kål) hvis
modsat(Bonde,Ulv_ged).

altså det er ulovligt, at ulven og geden er modsat bonden.
Lav selv den anden regel for en ulovlig tilstand.
Der findes 4 mulige træk, og vi beskriver det ene. Bonden sejler over alene:

træk(p(Gammel,Ulv,Ged,Kål),p(Ny,Ulv,Ged,Kål)) hvis
modsat(Gammel,Ny) og
ikke(ulovlig(Ny,Ulv,Ged,Kål)).

altså, et træk er at bonden tager alene til den anden bred, og at de kommer til en lovlig position. Prøv selv at skrive de resterende 3 træk op. Løsningen findes på SCANLOG-disketten under navnet **floden.** Bemærk, at du skal bruge reglerne for **udvid**, og at du skal stille spørgsmålet

find alle S så
udvid(<p(n,n,n,n)>,S,p(s,s,s,s)).

Reglerne for **udvid** kan hentes fra filen **udvid** på SCANLOG-disketten.

4. Prøv om du kan lave regler, som fortolker svaret du får fra løsningen på flodpassage-problemet. Reglerne kan laves, så SCANLOG til spørgsmålet

find alle Godt så
udvid(<p(n,n,n,n)>,S,p(s,s,s,s)) og
fortolk(S).

svarer

De er alle på nordbredden.
Bonden sejler med ...
.
.
.
De er alle på sydbredden.
Et svar er Godt

Se afsnit 5.13 (eller **skriv** og **nylinie** i opslagsværket bagest i bogen) for en beskrivelse af primitiver til skrivning på skærmen. Reglerne findes på filen **fortolk** på SCANLOG-disketten.

5. Illustrerende eksempler

I dette kapitel gennemgås en række illustrerende eksempler på anvendelsen af SCANLOG. Brug af **hvordan**-knappen kan evt. hjælpe til forståelsen af reglerne. Reglerne til de eksempler, der gennemgås, ligger alle på disketten.

5.1 Franske verber

I dette eksempel skal vi se på bøjningen af de regelmæssige (ikke refleksive) franske verber. Dette er blot en måde at gøre det på, men vi skal prøve at lave det så generelt, at det også kan bruges til bøjning af andre regelmæssige verber fra f. eks. spansk ved kun at ændre de størrelser, som er specielle for spansk (bøjningen og endelserne). Vi laver regler, så det bliver muligt at have nogle uregelmæssige verber med. Til sidst skal vi se på, hvordan vi kan ændre reglerne, så vi kan bøje verberne i mere end een tid.
De regelmæssige verber kan opdeles i tre grupper - hver med sin bøjning. Verbets navnemåde-endelse bestemmer bøjningsgruppen. Den enkelte gruppes bøjning er bestemt af tid, person og tal.
Når man bøjer verber bruges følgende fremgangsmåde:

1. Find ud af hvilken bøjningsgruppe verbet tilhører, og hvilket stamme verbet har.
2. Find ud af hvilken endelse, der skal på verberne i denne bøjningsgruppe, når vi kender tid, person og tal.
3. Den fundne endelse sættes på verbets stamme.

Regelmæssige verber, der ender på 'er' i navnemåde hører til 1. bøjningsgruppe. Verber der ender på 'ir' hører til 2. bøjningsgruppe, og verber med navnemåde-endelse 're' tilhører 3. bøjningsgruppe.
Vi laver regler, der giver os sammenhængen mellem en endelse og bøjningsgruppen.

```
endelse_gruppe('er',1).
endelse_gruppe('ir',2).
endelse_gruppe('re',3).
```

Vi skal have fundet ud af verbets type, og vi skal også have verbets stamme.
Ved hjælp af streng-primitivet **sammensæt** (se kapitel 2), ser vi på, hvordan de mulige endelser passer med verbet.

```
; Verbum skal være verbet i navnemåde.
; Stamme er resten af Verbum, når endelser er fjernet.
find_type(Verbum,Type,Stamme) hvis
    endelse_gruppe(Endelse,Type) og
    sammensæt(Stamme,Endelse,Verbum).
```

I ovenstående regel finder vi først de mulige endelser og sammenligner med **Verbum.** Hvis **Verbum** har en endelse, som er en af dem vi kender, så er **Type** bøjningsgruppen, og **Stamme** bliver det, der er tilbage, når **Endelse** er fjernet fra **Verbum.**

Vi prøver at stille nogle spørgsmål, for at se om reglen virker, som vi venter det.

```
kan find_type('vendre',3,'vend') bekræftes.
Ja
find alle Type og Stamme så
    find_type('parler',Type,Stamme).
Et svar er 1 og 'parl'
Ikke flere løsninger
find alle Verbum så find_type(Verbum,1,'all').
Et svar er 'aller'
Ikke flere løsninger
find alle Verber så find_type(Verber,Type,'fin').
Et svar er 'finer'
Et svar er 'finir'
Et svar er 'finre'
Ikke flere løsninger
```

Som det ses af de sidste spørgsmål, kan **find_type** sætte en endelse på en stamme, og dermed foreslå, hvad et verbum kan hedde i navnemåde.
Vi vil lave regler, der fortæller hvilken endelse, der skal på stammen for de tre bøjninger, som er vist i tabellen i figur 5.1. Vi skal for hver bøjning have en endelse, der er bestemt ud fra, hvilken person det er.

	1. bøjning	2. bøjning	3. bøjning
je	e	is	s
tu	es	is	s
il(elle)	e	it	
nous	ons	issons	ons
vous	ez	issent	ez
Ils(Elles)	ent	issent	ent

Figur 5.1 Verbernes endelse i de tre bøjningsgrupper.

```
bøjning(1,'je','e').
bøjning(1,'tu','es').
bøjning(1,'il','e').
bøjning(1,'elle','e').
bøjning(1,'nous','ons').
```

```
bøjning(1,'vous','ez').
bøjning(1,'Ils','ent').
bøjning(1,'Elles','ent').
bøjning(2,'je','is').
bøjning(2,'tu','is').
bøjning(2,'il','it').
bøjning(2,'elle','it').
bøjning(2,'nous','issons').
bøjning(2,'vous','issez').
bøjning(2,'Ils','issent').
bøjning(2,'Elles','issent').
bøjning(3,'je','s').
bøjning(3,'tu','s').
bøjning(3,'il',' ').
bøjning(3,'elle',' ').
bøjning(3,'nous','ons').
bøjning(3,'vous','ez').
bøjning(3,'Ils','ent').
bøjning(3,'Elles','ent').
```

Vi kan stille spørgsmål for at se, hvordan **bøjning** og **endelses_gruppe** arbejder sammen

```
find alle Stamme og Endelse så
    find_type('finir',Type,Stamme) og
    bøjning(Type,'nous',Endelse).
Et svar er 'fin' og 'issons'
Ikke flere løsninger
```

Som det ses virker de to regler sammen, fordi **endelse_gruppe** finder ud af, hvad **Type** skal være (her 2), og hvad **Stamme** kan være. **bøjning** kan så finde ud af, at endelsen skal være **'issons'.**
Vi har næsten lavet en regel, der kan bøje verber ud fra de to sæt regler. Vi mangler blot at sætte stammen og endelsen sammen.

```
bøj(Verbum,Person,Resultat) hvis
    find_type(Verbum,Type,Stamme),
    bøjning(Type,Person,Endelse) og
    sammensæt(Stamme,Endelse,Resultat).
```

Vi vil stille nogle spørgsmål med **bøj.**

```
kan bøj('chanter','tu','chantes') bekræftes.
Ja
find alle Resultat så bøj('chanter','nous',Resultat).
Et svar er 'chantons'
Ikke flere løsninger
```

```
find alle Person og Resultat så
    bøj('finir',Person,Resultat).
Et svar er 'je' og 'finis'
Et svar er 'tu' og 'finis'
Et svar er 'il' og finit'
Et svar er 'elle' og 'finit'
Et svar er 'nous' og 'finissons'
Et svar er 'vous' og 'finessez'
Et svar er 'Ils' og 'finissent'
Et svar er 'Elles' og 'finissent'
Ikke flere løsninger

find alle Verbum så bøj (Verbum,'je','finis').
Ingen løsninger
```

Som det ses af det sidste spørgsmål, så finder **bøj** ingen løsninger, når **Verbum** ikke kendes. Grunden til dette er, at

```
find_type(Verbum,Type,Stamme)
```

ikke finder løsninger. **find_type** skal have enten et verbum eller en stamme for at finde værdier til de resterende variable. Vi skal altså bruge en anden løsningsmetode til **bøj,** når vi ikke kender verbet.
Vi vil nu prøve at lave en lidt anderledes regel, så vi kan finde en person, og en mulig navnemåde af verbet, når vi har en bøjet form at det. Vi kan forsøge at bruge **bøjning** til at finde en mulig endelse (og person). Denne endelse kan vi bruge til at finde en stamme, og dernæst sætte den sammen med en af de endelser, et verbum kan have i navnemåde.

```
; Regel, der bruges til at finde navnemåde af et verbum,
; når Resultat er kendt, men Verbum ikke er kendt.
bøj(Verbum,Person,Resultat) hvis
    bøjning(Type,Person,Endelse),
    sammensæt(Stamme,Endelse,Resultat) og
    find_type(Verbum,Type,Stamme).
```

Vi spørger SCANLOG

```
find alle Verbum så bøj(Verbum,'je','parle').
Et svar er 'parler'
Ikke flere løsninger
find alle Verbum og Person så bøj(Verbum,Person,'finis').
Et svar er 'finir' og 'je'
Et svar er 'finir' og 'tu'
Et svar er 'finire' og 'je'
Et svar er 'finire' og 'tu'
Ikke flere løsninger
```

I det øverste spørgsmål finder SCANLOG først, at

```
bøjning(1,'je','e').
```

kan bruges, og en mulig stamme bliver der **'par1'. Type** blev af **bøjning** sat til 1, og derfor kommer endelsen **'er'** på stammen. I det sidste spørgsmål, fremkommer de to sidste løsninger, fordi den endelse man sætter på i 3. bøjning, også passer med **'finis'**.
Den sidste regel med **bøj** er ikke helt så gennemskuelig, som den første, men findes et bøjet verbum, vil SCANLOG kunne foreslå, hvad verbet hedder i navnemåde. (Der kan godt være flere muligheder). Prøv evt. at stille **find**-spørgsmål, og få dem uddybet med **Hvordan**-knappen.
Den sidste regel er mere langsom end den første, og den er kun nødvendig, når **Verbum** ikke kendes. For at sikre, at det altid er den rigtige af de to **bøj**-regler, der kommer i brug, forsyner vi dem med en **hvor**-del, så kun den første kommer i brug, når **Verbum** kendes, og den sidste kun når **Resultat** kendes. Vi sætter en **vælg_een** omkring dem, for det er kun nødvendigt, at een af reglerne bliver prøvet.

```
vælg_een bøj
   ; Regel, der bruges, når Verbum er kendt.
   bøj(Verbum,Person,Resultat) hvor
        streng(Verbum) hvis
        find_type(Verbum,Type,Stamme),
        bøjning(Type,Person,Endelse) og
        sammensæt(Stamme,Endelse,Resultat).

   ; Regel, der bruges til at finde navnemåde af et verbum,
   ; når Resultat er kendt, men Verbum ikke er kendt.
   bøj(Verbum,Person,Resultat) hvor
        streng(Resultat) hvis
        bøjning(Type,Person,Endelse),
        sammensæt(Stamme,Endelse,Resultat) og
        find_type(Verbum,Type,Stamme).
slut.
```

Du kan se reglerne i brug, ved at bruge **Hvordan**-knappen.

Verbernes bøjning bliver ikke altid rigtig, for det er kun et fåtal af de eksisterende regler om verbernes bøjning, som bliver brugt her. Der er også visse områder, hvor der ikke eksisterer regler. Her må man lære tingene udenad. F.eks. de uregelmæssige verber.
Alle bøjnings-reglerne til de franske verber ligger på SCANLOG-disketten under navnet **verber.**

Spanske verber
Hvis reglerne skal ændres, så de passer til spanske verber, skal man lave andre regler med **endelse_gruppe** og **bøjning.** Derefter kan man bruge de samme regler.

5.2 Opgaver

1. Find ud af hvad parler hedder i 3. person ental.
2. Hvilke mulige verber og personer vil du forvente, der vil komme ud af at spørge SCANLOG? (skriv ned på et stykke papir).

   find alle Person og Verbum så
   bøj(Verbum,Person,'finissons').

3. Hvad hedder être i 1. person flertal?
   Hvad foreslår SCANLOG?
4. Vi vil i de næste opgaver udvide reglerne, så de er forberedt til at klare endnu en tid (f. eks. imparfait eller futur).
   Vi skal derfor have flere bøjninger lagt ind (andre endelser). Udvid de eksisterende regler med **bøjning,** så den får fire parametre. Den nye parameter skal fortælle hvilken tid bøjningen gælder for.
   De nuværende regler med **bøjning** skal alle ændres til

   bøjning(present,1,'il','e').
   bøjning(present,1,'tu','es').

   Reglerne med **bøj** skal også ændres, så de får fire parametre. Det skal være tiden, som skal med, så man kalder **bøj** med

   bøj(Tid,Verbum,Person,Resultat)

   De steder hvor **bøj** bruger **bøjning,** skal der også ændres så **Tid** kommer med. Lav alle disse rettelser, og prøv nu om reglerne stadig kan bøje verber som før. Husk at spørge **bøj** med present.
5. Tilføj nu nogle regler til **bøjning,** så der kommer en ny tid med. F. eks. imparfait eller futur.
6. Vi kan lægge nogle af de uregelmæssige verber ind ved at lave flere **bøj**-regler. Et verbum som boire (at drikke) er i present regelmæssigt i ental, men ikke i flertal.
   Indsæt nu flg. regler før de eksisterende med **bøj**

   bøj(present,'boire','nous','bovons').
   bøj(present,'boire','vous','buvez').
   bøj(present,'boire','Ils','boivent').
   bøj(present,'boire','Elles','boivent').

   Find selv nogle gode egnede verber, som kan indsættes.
7. Læg être og avoir ind som uregelmæssigt verbum.
8. Prøv at lave **bøj**-regler, så der kan bøjes verber i en sammensat tid f. eks. passé composé.

5.3 Morse koder

Vi vil i dette eksempel se på kodning af en tekst, i dette tilfælde morsekoden. Vi behandler her morsekoden, men det er meget få ting, der skal ændres, hvis man vil bruge reglerne til en anden kode. Koder vil også blive brugt i eksemplet med DNA og proteinsyntese (afsnit 5.5).

Når man arbejder med morse, tager man et tegn, oversætter det til morse, skriver det ned, o.s.v. Vi vil i dette eksempel lave en regel, der kan oversætte en streng til den tilsvarende i morse.
Vi vil først indlægge informationer om koden. I morse svarer 'a' til '.-', 'b' til '-...', o.s.v. Vi vil anbringe en blank bagest i morsestrengen, for så kan vi se, hvornår et tegn er slut. En blank ' ' vil vi oversætte som en blank. Dette giver os følgende fakta.

```
kodning('a','.- ').
kodning('b','-... ').
kodning('c',
..
..
kodning(' ',' ').
```

Da de store og små bogstaver er ens i morsekoden, har vi udeladt de store.
Vi skal nu til at fortælle SCANLOG, hvordan man koder en streng til en anden streng. Vi bruger strengprimitivet **sammensæt**, som kan virke opdelende, til at dele strengen op med. Vi oversætter den del for del til morse, og til sidst sammensætter vi stumperne til en morsestreng. Dette udtrykker vi i denne regel:

```
; Reglen oversætter fra tekst til morse
kod(Tekst,Kode) hvis
    sammensæt(S1,S2,Tekst),
    kodning(S1,K1),
    kod(S2,K2) og
    sammensæt(K1,K2,Kode).
```

Vi kan prøve, hvordan reglen virker, ved at spørge SCANLOG

```
kan kod('abc','.- -... -.-. ') bekræftes.
Ja
find alle Morse så kod('abe',Morse).
Et svar er '.- -... . '
Ikke flere løsninger
find alle Tekst så kod(Tekst,'.- -... ').
Ingen løsninger
```

Vi fandt ingen løsninger på det sidste spørgsmål, så vi skal have en anden regel, når en morsekode skal dechifreres. Læg mærke til at der er en blank bagest i morsestrengen. Hvorfor er denne nødvendig?
Hvis vi skal oversætte fra morse til almindelig tekst, så arbejder vi som før, blot opdeler vi morsestrengen, dechifrerer tegn for tegn, og sammensætter den oprindelige streng. Vi får derfor denne regel

```
kod(Tekst,Kode) hvis
    sammensæt(K1,K2,Kode),
    kodning(S1,K1),
    kod(S2,K2) og
    sammensæt(S1,S2,Tekst).
```

Vi kan nu lave en hvordel på de to regler for dermed at sikre, at SCANLOG bruger den rigtige regel på det rigtige tidspunkt.
Vi kan også sætte en **vælg_een**-konstruktion omkring de to regler for at sikre, at højst een af dem bliver prøvet.
Vi kan dog gøre oversættelsen mere effektiv ved at sikre, at der kun kommer en kodning af hver tekst, for vi ved, at morsekoden er entydig, så der er ingen grund til at lede efter flere løsninger, når den første er fundet. Vi vil derfor sætte en **find_een** på **kod** (se evt. i kapitel 2 om **find_een**). Dette er gjort på de regler til morsekodning, som ligger på disketten i filen **morse.**
Vi kan undgå, at SCANLOG finder mere end eet kald af **kodning,** der går godt, ved at lægge en **vælg_een**-konstruktion omkring alle **kodning**-reglerne.
Reglerne til dette eksempel ligger på disketten under navnet **morse.**

5.4 Opgaver

I disse opgaver kan det være en fordel at bruge **husk** til at gemme en streng med, for så er man fri for at taste en kodet streng ind igen. f. eks.

kan kod('fritime',Kodet) og husk(tekst(Kodet)) bekræftes.

husk laver den regel, der står som argument, og ovenfor vil der blive lavet en regel med navnet **tekst.** Reglen har den kodede streng som argument. Du kan så senere spørge

find X så tekst(Kodet) og kod(X,Kodet).

1. Hvad bliver 'fritime' i morse?
2. Hvad står der her '-.- --- .-.. -.. - ---. .-.. ' i almindeligt sprog?
3. Prøv at slette **vælg_een**-konstruktion omkring **kodning,** og se hvor meget langsommere det går.
4. Hvad ville der ske, hvis vi slettede den blanke, der er efter hvert morsebogstav?
   Prøv at slette den ved f. eks. 'e' og 't', kod en streng og prøv at afkode den.
5. Lav din egen kode.

5.5 DNA og proteiner

I dette eksempel vil vi se på det område fra biologien. Eksemplet handler om proteinsyntesen; mere specielt om sammenhængen mellem de oplysninger, der er indkodet i DNA, og hvordan oplysningerne medfører en bestemt rækkefølge af aminosyrer i proteinsyntesen.
I eksemplet vil vi se hvilke DNA koder, der medfører bestemte rækkefølger af proteiner. Vi vil også se hvilke DNA koder, der kan give en bestemt rækkefølge af aminosyrer.
Når der skal laves protein, skal der først laves en skabelon fra DNA. Messenger-RNA (mRNA) er denne skabelon. mRNA transporteres så ud af cellenkernen. I cytoplasmaet kobler nogle transfer-RNA (tRNA) medbringende aminosyrer sig på mRNA. tRNA sætter sig på mRNA fra den ene ende, og det er så den aminosyre tRNA medbringer, som kommer med i proteinet. Se evt. figur 6.1 i bogen »Biologisk Forskning«.

Informationen i DNA og RNA er bestemt ud fra rækkefølgen af disse nukleotider (se tabellen i figur 5.2). Når der laves en

DNA		RNA	
Adenin	(a)	Adenin	(a)
Thyamin	(t)	Uracil	(u)
Guanin	(g)	Guanin	(g)
Cytosin	(c)	Cytosin	(c)

Figur 5.2 Nukleotider.

skabelon af DNA, så sker koblingerne som vist i tabellen i figur 5.3. Når der er lavet en skabelon af DNA til mRNA, og det er bragt ud af cellekernen, kan proteinsyntesen begynde. Syntesen starter i den ene ende og flytter sig hen ad mRNA'et.

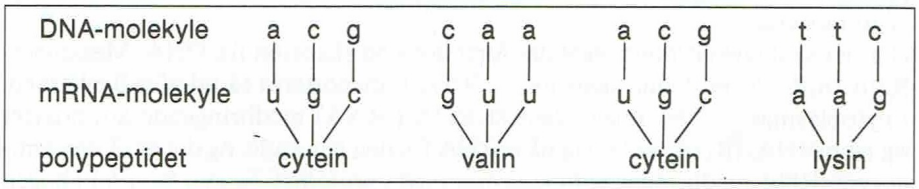
DNA	RNA
Adenin	Uracil
Thyamin	Adenin
Guanin	Cytosin
Cytosin	Guanin

Figur 5.3 Kobling mellem DNA og RNA.

Der er ca. tyve aminosyrer. Det er indkodet i par af tre mRNA-baser (en codon) hvilke aminosyrer, der skal sættes sammen. Man starter forrest i mRNA, og så tager man tre ad gangen. For hver tre RNA-baser kommer der en aminosyre mere på polypeptidet (proteinstumpen).
I tabellen i figur 5.4 kan du se RNA-koden for nogle enkelte aminosyrer. Vi kan illustrere processen som vist i figur 5.5.

RNA-kode	Aminosyre
ugc	cytein
guu	valin
aag	lysin
..	..

Figur 5.4 RNA-kode.



Figur 5.5.

Vi kan betragte dette problem som en art kodning, der svarer til den i afsnit 5.3. Vi vil nu indlægge oplysninger om kodningen fra DNA til mRNA, og kodningen fra de tre par af baser til aminosyrer.

```
; kodning fra DNA til RNA.
kodning('a','u').
kodning('t','a').
kodning('g','c').
kodning('c','g').
; kodning fra codoner til aminosyrer.
kodning1('gcu',alanin).
kodning1('ugc',cystein).
kodning1('guu',valin).
kodning1('aag',lysin).
..
```

Koderne kan ses i appendiks L i bogen »Biologisk Forskning«, men de er lagt ind som regler og kan indlæses fra filen **dna.** Der kan godt være flere forskellige koder for en aminosyre. Derfor må der ikke laves **vælg_een** omkring **kodning1**-reglerne.
Her er den regel, som vi vil bruge til at oversætte fra DNA- til RNA-kode.

```
kod(<>,<>).
kod(<H..T>,<H_kodet..T_kodet>) hvis
    kodning(H,H_kodet) og
    kod(T,T_kodet).
```

Der skal også bruges en tilsvarende regel **(kod1)** til at oversætte fra codoner til aminosyrer, men den er ikke vist her. Reglen bruger blot **kodning1** istedet for **kodning** (se evt. reglerne på disketten).
Vi skal også have lavet en regel, der kan samle tre baser til en streng på tre bogstaver, som vi har gjort i nedenstående regler.

```
find_codon(<>,<>).
find_codon(<S1,S2,S3..Rest>,<S..Rest_samlet>) hvor
    streng(S1),
    streng(S2) og
    streng(S3) hvis
    sammensæt(S1,S2,S_res),
    sammensæt(S_res,S3,S) og
    find_codon(Rest,Rest_samlet).
```

Til den regel, som skal finde rækkefølgen af aminosyrer, vil vi helst give en streng af nukleotider, men det er mest praktisk at arbejde med, hvis det er en liste af tegn, så vi laver en regel, der omformer mellem en streng og en liste af enkelttegn. Der er nedenfor vist et eksempel med reglen **omform.**

```
find X så omform('abcdef',X).
Et svar er <'a','b','c','d','e','f'>
```

Reglen er simpel og er med på disketten.

Vi skal til at lave en regel, der omdanner en liste af DNA-'bogstaver' til en liste af aminosyrer.

```
syntese(Læs_DNA,Aminosyrer) hvor
    streng(Læs_DNA) hvis
    omform(Læs_DNA,DNA_kode),
    kod(DNA_kode,RNA_kode),
    find_codon(RNA_kode,Codon_kode) og
    kod1(Codon_kode,Aminosyrer).
```

Vi kan nu stille spørgsmålet

```
find X så syntese('acgcaattc',X).
En løsning er <cystein,valin,lysin>
```

Det skal forstås sådan, at de ni nukleotider giver anledning til, at de tre aminosyrer skal sættes sammen i denne rækkefølge.
De regler, der skal bruges for at afprøve dette eksempel ligger på en fil med navnet **dna.** Reglerne for **syntese** og **find_codon** er udbygget, så der også kan stilles spørgsmål som

```
find alle X så syntese(X,<valin,lysin,alanin>).
```

5.6 Opgaver

Husk at den streng, der gives med til **syntese,** skal have en længde, der er delelig med 3.

1. På hvor mange måder kan man lave aminosyre-rækkefølgen lysin, valin, cystein, valin?
2. Prøv at sætte en ekstra DNA-nukleotid ind i en kode og udelad f. eks. den sidste. Prøv at se den forskel, der kommer blot et enkelt nukleotid bliver skubbet ind. Det er vist ret usandsynligt, at det sker (proteinet bliver et helt andet.)
3. Prøv at ændre et DNA-nukleotid til et andet, og se om det giver nogen forskel. Det kalder vi en enkelt mutation.
4. Prøv at se hvor stor forskel der er mellem koderne for en bestemt aminosyre. Er der nogen sammenhæng mellem de forskellige koder, så en enkelt mutation evt. giver den samme aminosyre?
   Hvis der er en sammenhæng, så prøv at finde en forklaring på det.
5. Hvorfor er der ikke bare een kode for een aminosyre?

5.7 Oversættelse af tidsangivelser

Vi vil oversætte nogle danske tidsangivelser til engelsk. Det gør vi ved først at oversætte dem til et 'fælles-sprog', og så senere oversætte 'fælles-sproget' til engelsk (se figur 5.6). Som 'fælles-sprog' vil vi bruge standardtid, d. v. s. 3 minutter over 2 vil vi oversætte til 2:03. I tabellen i figur 5.7 kan du se eksempler på danske tidsangivelser. Vi har isoleret de enkelte elementer i tidsangivelsen, så vi kan angive, hvad de danske tidsangivelser svarer til i fællessprog, se figur 5.8.

Vi har forsøgt det samme for engelske tidsangivelser (se tabellen i figur 5.9).

Figur 5.6 Oversættelse.

Nr.	Angivelse
1	3
2	3 21
3	halv 4
4	25 minutter over 4
5	10 minutter i 5
6	kvart over 5
7	kvart i 6
8	5 minutter i halv 7
9	4 minutter over halv 8

Figur 5.7 Danske tidsangivelser.

Nr.	Angivelse	Oversat
1	<timer>	<timer> : 00
2	<timer> <min>	<timer> : <min>
3	halv <timer>	<timer> -1 : 30
4	<min> minutter over <timer>	<timer> : <min>
5	<min> minutter i <timer>	<timer> -1 : 60-<min>
6	kvart over<timer>	<timer> : 15
7	kvart i <timer>	<timer> -1 : 45
8	<min> minutter i halv <timer>	<timer> : 30-<min>
9	<min> minutter over halv <timer>	<timer> -1 : 30+<min>

Figur 5.8 Oversættelse af danske tidsangivelser.

På engelsk er det ikke muligt at komme med en form, der svarer til nr. 8 og nr. 9 i den danske tidsangivelse. Vi har også her isoleret de enkelte dele i tidsangivel serne på engelsk (se figur 5.10).

Nr.	Angivelse
1	3 o'clock
2	3 21
3	half past 3
4	25 minutes past 4
5	10 minutes to 5
6	a quarter past 5
7	a quarter to 6

Figur 5.9 Engelsk tidsangivelse.

Nr.	Angivelse	Oversat
1	<timepart> o'clock	<timepart> : 00
2	<timepart> <minpart>	<timepart> : <minpart>
3	half past <timepart>	<timepart> -1 : 30
4	<minpart> minutes past <timepart>	<timepart> : <minpart>
5	<minpart> minutes to <timepart>	<timepart> -1 : <minpart>
6	a quarter past <timepart>	<timepart> : 15
7	a quarter to <timepart>	<timepart> -1 45

Figur 5.10 Oversættelse af engelske tidsangivelser.

Reglerne er lavet, så man kan spørge som vist nedenfor.

```
find alle Tid så
    dansk_stand('5 minutter i halv 4',Tid).
Et svar er <3,25>
Ikke flere løsninger
find alle Angivelse så
    dansk_stand(Angivelse,<2,30>).
Et svar er '2 30'
Et svar er 'halv 3'
Ikke flere løsninger
find alle Time så
    dansk_engelsk('6 minutter i 7',Time).
Et svar er '6 54'
Et svar er '6 minutes to 7'
Ikke flere løsninger
```

Der er tre regler, som man kan spørge på

dansk_stand Oversættelse mellem dansk og standardtid.
engelsk_stand Oversættelse mellem engelsk og standardtid.
dansk_engelsk Oversættelse mellem dansk og engelsk.

Reglerne ligger på disketten under navnet **tid.**
Brug **Hvordan**-knappen til at se reglerne i brug.

Socialministeriets bekendtgørelse nr. 443 af 6 juli om lov om børnetilskud og andre familieydelser.

Kapitel 1
Børnetilskud

Paragraf 1. Til børn under 18 år ydes børnetilskud og særligt børnetilskud efter reglerne i denne lov.

Paragraf 2. Børnetilskud udgør 1.168 kr. årlig for hvert barn.

Stk. 2. Forhøjet børnetilskud udgør 1708 kr. årlig pr. barn. Det ydes,

1) når den, der har forældremyndigheden over et barn, er enlig forsørger.

2) når begge et barns forældre modtager folkepension efter paragraf 3, stk. 1, i lov nr. 218 af 4. juni 1965 om folkepension eller invalidepension efter paragraf 3, stk. 2-4, i lov nr. 219 af 4. juni 1965 om invalidepension m. v. eller

3) når folke- eller invalidepension til én eller begge af to pensionsberettigede forældre er bortfaldet efter paragraf 26 i lov om folkepension eller i paragraf 31 i lov om invalidepension, eller når ophold på et alderdomshjem eller plejehjem for én af forældrene ifølge loven om omsorg for invalidepensionister træder i stedet for udbetaling af pensionen.

Paragraf 3. Den, der har forældremyndigheden over ét eller flere børn, som han har hos sig, og for hvilke der ydes forhøjet børnetilskud efter paragraf 2, stk. 2, nr. 1), er berettiget til ét ekstra børnetilskud, der udgør 1.228 kr. årlig.

Paragraf 4. Særligt børnetilskud ydes ud over børnetilskud i henhold til paragraf 2.

Stk. 2. Det særlige børnetilskud udgør 3.756 kr. årlig, når ingen af et barns forældre lever.

Stk. 3. Det særlige børnetilskud udgør 2.268 kr. årlig

1) til et barn uden for ægteskab, når faderskabet ikke er fastslået, og ingen er anset som bidragspligtig til barnet,

2) når kun én af et barns forældre lever,

3) til et barn, der, efter at forældremyndighedens indehaver er død, adopteres af dennes ægtefælle eller af en enlig person i barnets slægt,

4) når én eller begge et barns forældre modtager folkepension efter paragraf 3, stk. 1, i lov om folkepension eller invalidepension efter paragraf 3, stk. 2-3, i lov om invalidepension, eller

5) når folke- eller invalidepension til én af forældrene er bortfaldet efter paragraf 26 i lov om folkepension eller paragraf 31 i lov om invalidepension, eller når ophold på et alderdomshjem eller plejehjem for én af forældrene ifølge loven om omsorg for invalidepensionister og folkepensionister træder i stedet for udbetaling af pensionen.

Stk. 4. Ydes der ikke børnetilskud efter stk. 3, udgør det særlige børnetilskud halvdelen af det i stk. 3 nævnte beløb, når én af et barns forældre modtager invalidepension efter paragraf 3, stk. 4, i lov om invalidepension.

Figur 5.11 Børnetilskudsloven.

Lov om ændring af lov om børnetilskud og andre familieydelser og om ændring af lov om påligningen af indkomst- og formueskat til staten.

Paragraf 1

I lov om børnetilskud og andre familieydelser jfr. lovbekendtgørelse nr. 443 af 6. juni 1973 foretages følgende ændringer:

1 paragraf 1 affattes således:
"*Paragraf 1.* Til børn under 16 år ydes børnetilskud, og til børn under 18 år ydes særligt børnetilskud efter reglerne i denne lov."

2 paragraf 2, stk. 1, affattes således:
"Almindeligt børnetilskud udgør 1.568 kr. årlig for hvert barn."

3 paragraf 2, stk. 2 indledes således:
"Forhøjet børnetilskud udgør 2.361 kr. årlig for hvert barn. Forhøjet børnetilskud ydes istedet for almindeligt tilskud:".

4 I paragraf 3 ændres "1.228 kr." til: "1.804 kr.".

5 I paragraf 4, stk. 2, ændres "3.756 kr." til: "5.520 kr.".

6 I paragraf 4, stk. 3, ændres "2.268 kr." til: "3.336 kr."

Uddrag fra lov om invalidepension:

Kap. II
Ydelser til pensionister.

Paragraf 3. Retten til at få tilkendt invalidepension er betinget af, at erhvervsevnen på grund af fysisk eller psykisk invaliditet varigt er nedsat i det i stk. 2-4 angivne omfang.

Stk. 2 Personer, der må anses for erhvervsudygtige i ethvert erhverv, eller som kun har ubetydelig erhvervsevne i behold, er berettiget til invalidepension bestående af grundbeløb, invaliditetsbeløb samt beløb for erhvervsudygtighed.

Stk. 3 Personer, hvis erhvervsevne er nedsat i mindre grad end angivet i stk. 2, men dog med omkring 2/3, er berettiget til invalidepension bestående af grundbeløb og invaliditetsbeløb.

Stk. 4 Personer, hvis erhvervsevne er nedsat i mindre grad end angivet i stk. 3 angivne omfang, men dog mindst halvdelen, er berettiget til invalidepension bestående af halvt grundbeløb og halvt invaliditetsbeløb.

Figur 5.12 Rettelser og uddrag fra lov om invalidepension.

5.8 Beregning af børnetilskud

I dette eksempel har vi prøvet at lave regler, der ud fra børnetilskudsloven beregner hvor mange penge, der kan gives, når vi har oplysninger om barnets forhold (som det fungerer i dag). Den lovtekst vi er gået ud fra, er vist i figur 5.11 og 5.12. Det er vores lovfortolkning, vi har formaliseret, og det er ikke nødvendigvis den korrekte.

Sammen med reglerne til beregningen er der nogle regler, som kan bruges til at indsamle oplysninger om en person, så det bliver muligt selv at lave nogle specielle tilfælde.

Der er ydermere oplysninger om tre fiktive personer:

anne_line
siigurd
ingeborg

Når reglerne er indlæst fra filen **penge,** så kan der spørges på de tre personer. F.eks.

find Resultat så børnetilskud(anne_line,Resultat).

Ved at bruge **hvordan**-knappen er det muligt at få uddybet, hvorfor **anne_line** får det beløb, hun nu gør. Du kan se hvilke oplysninger, der er om **anne_line** sidst i regelsamlingen. Hvordan kan det være, at personen **ingeborg** kan få flere forskellige beløb? Skyldes det vores tolkning af loven, eller er loven tvetydig?

Oplysninger om en ny person kan opsamles ved at spørge

kan ny_person bekræftes.

Her skal alle svar skrives med små bogstaver, og når et svar er indtastet, skal der afsluttes med **Tegn Ind.**

5.9 Symbolsk differentiation

Som du måske har bemærket, bruger vi **værdi_af,** hvis vi ønsker at udregne værdien af et aritmetisk udtryk. Det gør vi, fordi SCANLOG er et symbolmanipulerende sprog, d.v.s. udtryk som f.eks. 3+X rent faktisk opfattes som 3+X og ikke som værdien 3+X. Det betyder, at vi kan manipulere med udtryk. Det skal vi udnytte til symbolsk differentiation.

Lad os betragte reglerne for differentiation *(k* betyder konstant, *U* og *V* er udtryk) i figur 5.13. På datamaskinen kan vi ikke skrive U^c, så vi indfører i stedet for funktionen **pot,** U^c = pot(U,c). SCANLOG ved ikke, hvad **pot** betyder. Vi laver *symbolsk* differentiation.

Reglerne for differentiation kan i SCANLOG skrives som følger:

```
vælg_een d
  d(K,X,0) hvor
       K <> X og
       (heltal(K) eller konstant(K)).
  d(X,X,1).
  d(-U,X,-A) hvis
       d(U,X,A).
```

Udtryk	Differentieret udtryk
dk/dx	*0*
dx/dx	*1*
d(-U)/dx	*-(dU/dx)*
d(U+V)/dx	*dU/dx + dV/dx*
d(U-V)/dx	*dU/dx - dV/dx*
*d(k*U)/dx*	*k*dU/dx*
*d(U*V)/dx*	*U*dV/dx + V*dU/dx*
d(U/V)/dx	*d(U*V^{-1})/dx*
d(U^{k})/dx	*k*U^{k-1}dU/dx*
d(ln(U))/dx	*U^{-1}dU/dx*

Figur 5.13 Differentiation.

```
d(U+V,X,A+B) hvis
        d(U,X,A) og
        d(V,X,B).
d(U-V,X,A-B) hvis
        d(U,X,A) og
        d(V,X,B).
d(K*U,X,K*A) hvor
        (konstant(K) eller heltal(K)) hvis
        d(U,X,A).
d(U*V,X,B*U+A*V) hvis
        d(U,X,A) og
        d(V,X,B).
d(U/V,X,A) hvis
        d(U*pot(V,-1),X,A).
d(pot(U,K),X,K*pot(U,K-1)*W) hvor
        (heltal(K) eller konstant(K)) hvis
        (K <> X og
        d(U,X,W).
d(ln(U),X,pot(U,-1)*A) hvis
        d(U,X,A).
slut.
```

Ved at bruge **vælg_een** sikrer vi os, at de korrekte regler anvendes i tvivlstilfælde, samt at SCANLOG ikke bruger tid på at søge efter flere ikke eksisterende løsninger.

Spørgsmål kan stilles på formen

```
find alle Diff så d(7*x+3*x+7,x,Diff).
Et svar er 7*1+3*1+0
Ikke flere løsninger
```

Man angiver altså det udtryk, som skal differentieres, derefter hvilken variabel, der differentieres med hensyn til, og til sidst hvor svaret skal leveres.

Prøv selv at finde ud af hvad SCANLOG vil svare til dette spørgsmål

find Diff så d(x*y+2*y*y+7*x,y,Diff).

Man kan også lave forenkling af udtrykkene, men det er et større arbejde. På disketten findes en regelsamling, der forenkler en del af de differentierede udtryk. Reglerne er ikke fuldstændige, men de kan dog reducere udtrykkene en del. Man kan derefter stille spørgsmål som

find Res så d(7*x+3*x+3,x,Diff) og simplificer (Diff,Res).
Et svar er 10

Reglerne til differentiation ligger på disketten under navnet **diff,** og reglerne til forenkling *(simplificering)* ligger under navnet **simpl.** Der er lidt mere om forenkling i det næste afsnit.

5.10 Forenkling af udtryk

På disketten findes filen **simpl,** som indeholder regler til forenkling af udtryk. Forenkling er ikke nogen simpel sag, men ved hjælp af **Hvordan-**knappen skulle der være gode muligheder for at forstå og eventuelt forbedre reglerne. Man kan f.eks. sørge for, at reglerne også kan klare udtryk, som indeholder parenteser. (Vi gør opmærksom på, at reglerne er afprøvet, men der kan måske forekomme mærkelige ting. I et sådant tilfælde er der kun en ting at gøre: find selv fejlen. Se også afsnittet om aritmetik i opslagsværket bagest i bogen). Man kan stille spørgsmål på formen

find Svar så simplificer(2*a*4*b+2-5,Svar).
Et svar er a*b*8-3

5.11 Tilfældigtalsgenerator

Vi vil nu have en regel, der giver os nogle forskellige tal, hver gang den bruges. Det er det, vi kalder »tilfældige« tal. For at det ikke skal være det samme tal, der fremkommer hver gang, vi bruger en regel, vil vi bruge et tal (en generator), som vi ændrer til næste gang, der skal laves et nyt tal.
Vi laver en regel, som vi kalder **generator.** Denne generator har kun een parameter, som skal være et tal mellem 1 og 100.

Når vi skal lave et tilfældigt tal, skal vi først finde den sidst dannede generator, dernæst skal vi udregne det »tilfældige« tal og en ny generator, og til sidst skal vi gemme den nye generator.

Se nu denne regel:

```
; Reglen danner pseudotilfældige tal.
; Det forudsættes, at der en en generator regel.
tilf_tal(Tilf) hvis
    glem(generator(X)),
    Tilf=værdi_af(((42*X) mod 101) - 1),
    Ny_X=værdi_af((42*X) mod 101) og
    husk(generator(Ny_X)).
```

Den aritmetiske funktion **mod** er brugt ovenfor: **X mod Y** giver resten ved heltalsdivision af **X** med **Y**, d. v. s. et tal mellem 0 og Y-1.
Vi bruger primitivet **glem** til at hindre, at der kommer mange **generator**-regler. **glem** sletter den første regel, der kan passe på indholdet.

```
glem(generator(X))
```

vil fjerne den første **generator**-regel, som har et argument (og **X** vil blive sat til, hvad dette argument var).
husk er et primitiv, som gør at argumentet bliver gemt som en regel.

SCANLOG regner kun med heltal mellem -32768 og 32767. Vær derfor opmærksom på at ingen udregning bliver større end 32767 eller mindre end -32768 (heller ikke noget mellemresultat), for da kan man ikke regne med resultatet.

5.12 Resumé

husk
Indsætter reglen, som er argument, i regel-samlingen.

glem
Sletter den første regel, som passer med argumentet. I reglerne til generering af tilfældige tal så vi, hvordan man kan bruge **husk** og **glem** til at gemme information mellem spørgsmål.

5.13 Høj-Lav spil

Spillet går ud på, at spilleren skal gætte et tal, som maskinen tænker på. Efter hvert gæt får spilleren at vide, om tallet var for højt eller for lavt. I vores spil skal der gættes et tal mellem 0 og 99.
Spillet er let at spille, men de anvendte regler er ikke helt enkle.
Vi vil først vise et eksempel, hvordan spillet foregår.

```
kan høj_lav bekræftes.
Høj-Lav spil. Gæt et tal på færrest forsøg.
Et tal mellem 0 og 99 (eller stop)? 50
```

```
For højt
Et tal mellem 0 og 99 (eller stop)? 28
For højt
Et tal mellem 0 og 99 (eller stop)? 3
For lavt
Et tal mellem 0 og 99 (eller stop)? 14
Du gættede det på 4 gæt
Ja
```

I spillet skal vi bruge noget, der tæller antallet af brugte forsøg; hertil kan vi bruge nedenstående **int**-regler. Hvordan de virker er henlagt som en opgave.

```
int(0).
int(N) hvis
    int(M) og
    N=værdi_af(M+1).
```

Til spillet skal vi også bruge tilfældigtalsgeneratoren fra afsnit 5.11.
Selve spillet skal først finde et tilfældigt tal, og derefter skal DANLOG blive ved at spørge spilleren om et tal, til det gættes. Når tallet er gættet stopper spillet.

Vi laver først denne regel

```
høj_lav hvis
    skriv('Høj-Lav spil. Gæt et tal på færrest forsøg.'),
    nylinie,
    tilf_tal(KODE) og
    find_een(
        int(Antal gæt),
        skriv('Et tal mellem 0 og 99 (eller stop ) ? '),
        læs(Gæt) og
        test_gæt(KODE,Gæt,Antal_gæt)
        ).
```

Primitivet **læs,** læser ind fra tastaturet til den variabel, der er som argument, og når spilleren har skrevet et tal, så accepteres der med **Tegn Ind. find_een**-konstruktionen er med, for når vi har fundet een løsning, skal vi ikke finde flere. Det er tænkt sådan, at **test_gæt** skal gå godt, hvis **KODE** og **Gæt** er ens, (eller spilleren skriver **stop),** og reglen skal ikke gå godt, hvis tallet er mindre end eller større end. Hvis spilleren gætter galt, da vil **test_gæt** fejle, og **int** vil foreslå et nyt tal, (**Antal_gæt** er een højere end før), spilleren spørges om et nyt gæt, o. s. v.
int vil blive ved med at lave løsninger, og **test_gæt** vil blive ved med at fejle, indtil tallet er gættet (eller der er skrevet **stop**).
I nedenstående regler bruges primitivet **falsk,** som aldrig kan blive sand.

Reglerne for **test_gæt** er

```
vælg_een test_gæt
  test_gæt(Kode,Gæt,Gange) hvor
      heltal(Gæt) og
      Kode=Gæt hvis
```

```
        Z=værdi_af(Gange+1),
        skriv('Du gættede det på ',Z,' gæt') og
        nylinie.
    test_gæt(Kode,Gæt,Gange) hvor
        heltal(Gæt) og
        Kode<Gæt hvis
        skriv('For højt)',
        nylinie og
        falsk.
    test<gæt(Kode,Gæt,Gange) hvor
        heltal(Gæt) og
        Kode>Gæt hvis
        skriv('For lavt'),
        nylinie og
        falsk.
    test_gæt(Kode,Gæt,Gange) hvor
        Gæt=stop hvis
        skriv('Tallet var ',Kode).
slut.
```

Alle reglerne ligger på disketten under navnet **spil.**

5.14 Resumé

læs
Der læses et argument fra tastaturet. Når dette er indtastet, skal der accepteres med **Tegn Ind.**

skriv
Argumenterne skrives ud på skærmen.

nylinie
Der laves et linieskift på skærmen.

5.15 Opgaver

1. Find ud af hvad **int** reglerne laver. Stil evt. spørgsmålet

   ```
   find alle X så int(X).
   ```

   Du kan afbryde med Ctl. c, d.v.s. du skal holde tasten, der hedder Ctl. nede, og så trykke på c-tasten.
2. Lav spillet, så det ikke kører frem og tilbage mellem **int** og **test_gæt,** men så **test_gæt** bliver rekursiv. (En svær opgave).

6. Mulige projekter

Vi vil til sidst give forslag til nogle projekter, som man kan lave i SCANLOG. De fleste af forslagene er tidligere lavet i logik-programmerings-sprog, bl.a. er SCANLOG blevet brugt til at lave semantisk analyse af sætninger.

- I matematik kan SCANLOG bruges til at vise, hvordan man regner med grupper og ringe.
- Der kan laves et projekt om syntaktisk analyse af simple sætninger ud fra en grammatik. Fastlæggelsen af denne grammatik er også en god opgave, idet man skal vide, hvordan vi opbygger sætninger.
- Et andet sprogprojekt er semantisk sætnings-analyse, og en evt. oversættelse af disse til f.eks. tysk.
- I biologi kunne man lave et system til artsbestemmelse af f.eks. svampe.
- SCANLOG kan også bruges til at undersøge simple kemiske reaktioner.
- Et andet projekt indenfor kemi er, at bruge SCANLOG i forbindelse med organisk analyse.
- I geografi bruger man modeller af netværk til trafikanalyse. Det kan let laves i SCANLOG, og man kan lave en analyse af dette netværk.

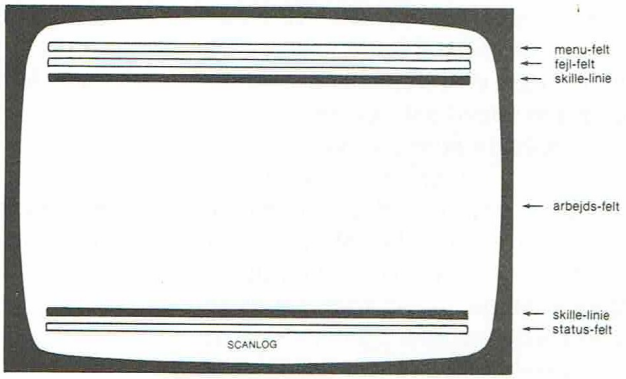
2. DEL

7. Opslagsværk

Dette kapitel er et opslagsværk til SCANLOG. Kapitlet indeholder en beskrivelse af alle kommandoer i SCANLOG, sprogets opbygning og hvilke primitiver, der er til rådighed i sproget.
SCANLOG består af en logik-sprogs-fortolker og et redigeringssystem. Fortolkeren er i stand til at løse problemer beskrevet ved regler, som er indtastet ved hjælp af redigeringssystemet. Fortolkeren kan desuden fortælle, **hvordan** en løsning til et problem er fundet. SCANLOG's redigeringssystem gør det muligt at indsætte regler, rette i dem, flytte dem m.m. Alt dette foregår ved en simpel regel-udpegning med markøren og aktivering af en SCANLOG kommando. Kommandoerne er beskrevet i en menu øverst på skærmen og aktiveres ved et enkelt tryk på en funktionstast på tastaturet. Ud over det findes der kommandoer til at skrive og læse regler på disketten, optage skærmdialog på en fil og meget mere. SCANLOG tilbyder hjælp ved fejl og mulighed for f.eks. at ændre i et tidligere stillet spørgsmål. Desuden viser SCANLOG hele tiden, hvad der sker. Det gøres ved hjælp af en statuslinie på skærmen samt anvendelse af bevægelige objekter på skærmen under mere tidskrævende operationer.

7.1 Skærmen

Skærmen er delt op i fire felter, som hver har en speciel funktion. De fire felter er (se figur 7.1)



Figur 7.1 Skærmbilledet.

menulinien Denne linie viser, hvilke kommandoer der på et givet tidspunkt er til rådighed, samt hvilken tast man skal trykke på for at aktivere kommandoen. Hvis der er flere kommandoer, end der er plads til på een linie, kan man bladre i menulinierne ved at trykke på mellemrumstangenten. Linien bruges også til indtastning af argumenter til nogle af SCANLOG's kommandoer.

statuslinien Linien viser, hvilken tilstand SCANLOG befinder sig i (hvad SCANLOG laver). Linien kan for eksempel vise, hvilken kommando man er ved at udføre, eller at SCANLOG venter på at komme til at bruge disketten.

arbejdsfelt Feltet bruges til at redigere, liste regler, stille spørgsmål, udskrive svar o. s. v.

meddelelsesfelt Linien bruges til udskrivning af fejlmeddelelser og andre korte meddelelser fra SCANLOG. Den bruges også til at markere aktivitet under tidskrævende operationer.

7.1.1 Indtastning og rettelse i arbejdsfeltet

Indtastning af regler eller spørgsmål i arbejdsfeltet foregår indenfor een regel eller spørgsmål. Under indtastningen er det muligt af flytte markøren mellem det første og det sidste bogstav i teksten, der indtastes. Et forsøg på at gå uden for dette område resulterer i et bip fra datamaskinen. Markøren kan flyttes v. h. a. ←, →, ↑, ↓ og \, hvoraf den sidste flytter markøren til det første bogstav i den indtastede tekst. Ved tryk på en bogstavtast indsættes bogstavet mellem tegnet til venstre for markøren og tegnet under markøren. Det er muligt at fjerne tegn v. h. a. ←, som fjerner tegnet til venstre for markøren, eller **Slet Tegn,** som fjerner tegnet under markøren. Der skiftes linie ved at trykke på ↵. En linie kan højst være 79 tegn lang; et forsøg på at gøre den længere resulterer i et bip fra datamaskinen. Der kan maksimalt bruges 19 linier til en regel eller et spørgsmål. Under indtastningen kan tasten **ESC** bruges til at slette hele den indtastede tekst (se menulinien på skærmen). Når en regel er færdigindtastet og accepteret af SCANLOG frigives de brugte linier igen. En regel accepteres af SCANLOG ved at trykke på **Tegn Ind.**

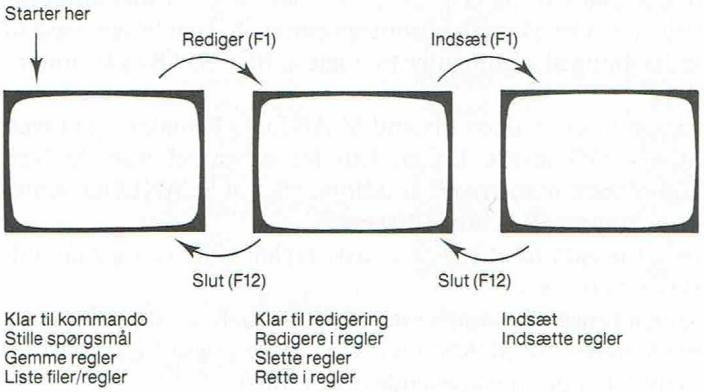
7.1.2 Indtastning af argumenter til kommandoer

Kommandoer, der kræver argumenter, benytter menufeltet til indtastning. Hvis SCANLOG selv foreslår et argument, kan dette accepteres ved at trykke på **Tegn Ind** eller ↵, ellers skriver man et andet argument. Det er ikke nødvendigt at slette SCANLOG's forslag, det sker automatisk, når man begynder indtastning af et andet argument. Hvis man har fortrudt aktiveringen af kommandoen, kan den afbrydes ved at give SCANLOG et tomt argument (hvis SCANLOG selv har foreslået et argument, kan det slettes ved at trykke på ←), d. v. s. ingen tekst, eller ved at trykke på **ESC**-tasten.

7.2 Systemet

SCANLOG består af to delsystemer, som kan betegnes med to tilstande, »Klar

til kommando« og »Klar til redigering« (se figur 7.2 som er en udvidelse af figur 1.3). I hver af disse tilstande findes en mængde kommandoer, som kan afgives. De forskellige tilstande er nærmere beskrevet i de efterfølgende afsnit.



Figur 7.2 Tilstande i SCANLOG.

7.2.1 Klar til kommando

SCANLOG starter i kommandotilstand, hvor man kan stille spørgsmål m.m. Tilstanden er markeret i statuslinien som »Klar til kommando«.

kan bekræftes SCANLOG sætter en skabelon op til et spørgsmål, hvortil der svares Ja eller Nej. Spørgsmålet stilles så ved at fylde skabelonen ud og få SCANLOG til at acceptere spørgsmålet. Se også kapitel 1.

find så SCANLOG sætter en skabelon op. Til spørgsmål af denne type finder SCANLOG een løsning, rapporterer den og giver derefter mulighed for uddybning af løsningen, flere løsninger m.m. Se kapitel 1 og 2 samt afsnit 7.2.4.

find alle så Som **find så** bortset fra at alle løsningerne udskrives uden pause, og der er ikke mulighed for uddybning. Se kapitel 1.

Gentag Det sidst accepterede spørgsmål bliver skrevet ud på skærmen, hvorefter det kan rettes eller ændres og prøves igen.

Rediger SCANLOG bringes i redigeringstilstand, hvor man kan redigere i sine regler (se afsnit 7.2.2).

List SCANLOG tilbyder at liste tre forskellige ting.
Man kan angive argumenterne.

- **alt** Alle regler bliver listet på skærmen.
- »navn« Alle regler med hoved svarende til navnet bliver listet på skærmen.
- **filer** Alle SCANLOG-filer på disketten bliver listet på skærmen. SCANLOG vil anmode om at få opgivet en diskettestation.

Listninger kan afbrydes, standses midlertidigt m.m. Se menulinien.

Læs fil

Der indlæses regler fra en fil. Der kan angives en anden diskettestation ved at skrive

<diskettestation>:<fil>

f.eks. a:floden.

Reglerne bliver indsat sidst i rækken af regler. Filnavnet får automatisk tilføjelsen **.scl.** Hvis man laver sine regler i et andet redigeringssystem, skal man være opmærksom på, at der *skal* være punktum efter hver regel, og at dette punktum *skal* være det sidste tegn på linien. Man skal også være opmærksom på, at en regel maksimalt må fylde 19 linier, som maksimalt kan indeholde 79 tegn hver.

Gem på fil

Kommandoen giver mulighed for at gemme regler på en diskette eller at få en udskrift af reglerne på en tilknyttet printer. Filnavnet får automatisk tilføjelsen **.scl.**

OBS: Printerudskrift opnås ved at angive filnavnet **papir.** Ved udskrift på printer *skal* der være en printer tilsluttet maskinen.

Diskette

Ved hjælp af denne kommando kan man angive hvilken diskettestation, SCANLOG skal læse fra eller skrive på for eftertiden.

Fjern

SCANLOG tilbyder at fjerne tre forskellige ting. Man kan som argument angive

alt Alle regler i SCANLOG fjernes, registerne tømmes og det sidst stillede spørgsmål fjernes (man kan derfor ikke benytte kommandoen **Gentag** umiddelbart efter brug af **Fjern alt**). Kommandoen frigiver samtidigt al lagerplads i SCANLOG og kan derfor eventuelt hjælpe ved lagermangel.

»navn« Alle regler med samme hoved, som det angivne navn, fjernes.

fil Gør det muligt at fjerne en fil. SCANLOG anmoder om et filnavn, og hvis filen findes på disketten, bliver den fjernet.

Dialog

Kommandoen kan bruges til at starte eller afslutte skrivning på en fil, som indeholder det, der bliver skrevet på skærmen. Når SCANLOG standses vil der være en fil ved navnet **dialog. txt,** som indeholder dialogen. Når dialogskrivning starter overskrives en eventuelt tidligere dialogfil. Hvis en dialog er skrevet, vil SCANLOG tilbyde udskrivning på printer ved afslutning af programmet.

OBS: Når der skrives dialog, er diskettestationen permanent reserveret, indtil dialogskrivning afbrydes igen. Dialogskrivning bør derfor kun benyttes på enkeltbrugersystemer eller på et tidspunkt, hvor der kun er een bruger på systemet.

Slut

Afslutter SCANLOG programmet. Inden programmet standser skal man bekræfte, at man ønsker at slutte. Hvis der er blevet skrevet en dialogfil tilbydes en udskrift af denne.

OBS: Der *skal* være tilsluttet en printer til datamaskinen, hvis en udskrift af dialogfilen vælges.

7.2.2 Klar til redigering

Denne tilstand nås fra kommandotilstand ved at afgive kommandoen **Rediger,** som starter SCANLOG's redigeringssystem. Her er det muligt at indsætte, rette, slette, indlæse regler, m.m.

Indsæt Der gøres klar til at indsætte nye regler, før reglen under markøren. Se f.eks. kapitel 1. SCANLOG skifter tilstand til indsættetilstand, som vist på figur 7.2. I menulinien kan man se de kommandoer, der kan afgives under indtastning af regler. Indtastning foregår som beskrevet i afsnit 7.1.1. Se desuden afsnit 7.2.3 for specielle kommandoer til kopiering af regler m.m.

Rette Kommandoen gør det muligt at rette i reglen under markøren. Man kan slette bogstaver, indsætte o.s.v. (se afsnit 7.1.1). Hvis man fortryder rettelsen, kan man bruge kommandoen **Fortryd,** som igen skriver den gamle regel ud på skærmen, og SCANLOG vender tilbage til redigeringstilstand. Det er ikke tilladt at rette i ordene **vælg_een** og **slut.**

Læs fil Virker som **Læs fil** i kommandotilstand (side 71), blot indsættes reglerne før reglen under markøren. Efter endt indlæsning placeres markøren ved den først indlæste regel.

Gem på fil Virker som **Gem på fil** i kommandotilstand (se side 71).

Slet Reglen under markøren fjernes. Reglen lægges i første omgang over i register nummer 0, og man kan indsætte den igen et andet sted ved at bruge **Flyt**-kommandoen. Se også afsnit 7.2.5.

Fjern Virker som **Fjern** i kommandotilstand (se side 71).

Vis Viser indholdet af et register på skærmen.

Flyt Ved hjælp af denne kommando kan man flytte regler **til** eller **fra** registre. Det er ikke muligt at flytte noget til register 0, som bruges til regler, der er fjernet med **Slet.** SCANLOG spørger, om man vil flytte **til** eller **fra** register, og hvilket register man vil bruge. Hvis registeret, der flyttes **til,** er i brug i forvejen, skal man bekræfte, at reglen i registeret kan fjernes. Se også afsnit 7.2.5.

Lav vælg_een Der laves en **vælg_een**-konstruktion omkring nogle regler. **vælg_een** indsættes der, hvor markøren er, markøren flyttes ned med markørpilene, og **slut** indsættes ved at trykke på **s.** Denne kommando er den eneste mulighed for at lave **vælg_een** omkring *eksisterende* regler. Det er muligt at undslippe fra kommandoen uden at lave **vælg_een** ved at indsætte **slut** umiddelbart efter **vælg_een.**

Nyt navn Kommandoen bruges til at ændre navn på en relation. Alle forekomster af det oprindelige navn vil fremover være erstattet af det nye. Det er ikke muligt at ændre til et navn, som allerede eksisterer (eller har eksisteret) i systemet. SCANLOG vil anmode om nyt og gammelt navn.

Slut Redigeringen afsluttes, og SCANLOG vender tilbage til kommandotilstand. Se figur 7.2.

Frem Bruges til enten at søge efter en regel med et bestemt hoved, eller til at hoppe et vist antal regler frem. SCANLOG anmoder om et argument, og man kan angive

»navn« SCANLOG søger frem til den første regel, som har hoved svarende til det angivne navn. Markøren placeres over denne regel.

»tal« SCANLOG hopper tal regler frem og placerer markøren over reglen.

Tilbage Som **Frem** blot flyttes markøren her tilbage.

Top Markøren flyttes til den første regel i samlingen af regler.

Bund Markøren flyttes til den sidste regel i samlingen af regler.

↑ Markøren flyttes en regel eller kommentar op.

↓ Markøren flyttes en regel eller kommentar ned.

7.2.3 Indsættelsestilstand

Ved at afgive kommandoen **Indsæt** kan man indsætte regler. Nedenstående kommandoer kan bruges på passende steder under indsættelsen (se i menulinien).

Kopi Kommandoen kan bruges, hvis man endnu ikke har skrevet noget, og man gerne vil have en regel, der minder om den, der står ovenover. **Kopi** laver en nøjagtig kopi, og man kan rette i den, før man accepterer.

Slut Man er færdig med at indsætte, og kan gå tilbage til redigeringstilstand. Markøren placeres over den sidst indsatte regel. Se figur 7.2.

Fortryd Teksten, der er skrevet siden sidste gang, man fik accepteret, bliver slettet fra skærmen.

Accepter SCANLOG undersøger om den forstår, hvad der er skrevet. Hvis det indtastede ikke er skrevet korrekt, rapporterer SCANLOG en fejl, Markøren viser stedet, hvor fejlen er opstået, og en fejlmeddelelse siger, hvad der er galt. Hvis man ikke kan finde fejlen, kan SCANLOG tilbyde hjælp, som udskrives på skærmen. Hjælpen giver eksempler på, hvordan fejlen kan være opstået (se menulinien for aktivering af hjælp).

7.2.4 Uddybningstilstand

Uddybningstilstand opnår man, når der er fundet en løsning til et **find så** spørgsmål (se også kapitel 2). Delspørgsmål, som indeholder **ikke** og **for_alle gælder** kan ikke uddybes (se afsnit 7.3.7). Kommandoerne er:

Fortsæt Der søges efter endnu en løsning.

Stop	Afbryd søgningen efter flere løsninger.
Alle	Udskriv alle de resterende løsninger uden mulighed for uddybning.
Top	Start uddybningen fra det stillede spørgsmål.
Op	Går tilbage til det spørgsmål, hvorfra man netop har fået uddybet et delspørgsmål.
Hvordan	Uddybning af et delspørgsmål. Hvis der er flere, anmoder SCANLOG om hvilket delspørgsmål, man ønsker uddybet.

7.2.5 Registre

Det hænder, at man har behov for at ændre rækkefølgen af regler eller midlertidigt fjerne en regel, mens man prøver et spørgsmål. Til dette formål har SCANLOG fem registre, som er et sted, hvor en regel kan opbevares. Man kan flytte **til** eller **fra** et register ved hjælp af kommandoen **Flyt.** Reglerne i registrene anvendes *ikke* under løsning af et spørgsmål. Registrene er nummererede fra 1 til 5. Udover disse findes også register nummer 0. Dette bruges af SCANLOG ved aktivering af kommandoen **slet.** Reglen, der slettes, flyttes derved over i register 0, og en eventuel gammel regel i dette register fjernes. Det er muligt at flytte **fra** register 0, så man kan redde en regel, som blev slettet ved en fejltagelse. Ved brug af kommandoen **Fjern alt** tømmes også registrene.

7.2.6 Fejlmeddelelser under listning, redigering og uddybning

Da SCANLOG bestemmer, hvordan en regel skal skrives på skærmen, kan en regel komme til at fylde mere end 19 linier (sker meget sjældent), og SCANLOG kan ikke vise reglen. I stedet for reglen udskrives teksten

»Her er en regel, der ikke er plads til ...«

og denne tekst markerer tilstedeværelsen af en regel, som er for stor til redigering. Det er muligt at fortsætte redigeringen, blot man ikke forsøger at rette i reglen. Reglen kan ses ved at bruge **List**-kommandoen, og den kan udskrives på fil. Den kan *ikke* indlæses fra fil igen, og det er altså nødvendigt at bruge et almindeligt redigeringssystem og dele reglen op i to eller flere delregler.

Under alle former for listning kan teksten »%MANGLER%« fremkomme i en regel. Dette skyldes lagermangel i SCANLOG og betyder, at SCANLOG ikke har lager nok til at udskrive reglen korrekt. Hvis det sker under uddybning af svar eller rapportering af løsning, er fejlen ikke så alvorlig. SCANLOG vil efter afbrydelse af spørgsmålet få mere lager til rådighed, og reglen kan derefter listes korrekt. Hvis fejlen forekommer under redigering eller listning, kan nogle af reglerne simpelthen ikke gemmes eller listes korrekt. Hvis man har reglerne gemt på en fil, kan man prøve at fjerne alle regler i SCANLOG og indlæse dem fra filen igen. Det skulle klare problemerne i de fleste tilfælde. Hvis man ikke har fået gemt reglerne, må man acceptere, at SCANLOG laver fejl under udskrivningen, og man skal rette reglerne bagefter i et almindeligt redigeringssystem.

Ved udskrivning af svar, uddybning eller brug af **skriv** vil SCANLOG benytte navnet på en variabel, hvis den ikke er bundet. Imidlertid kan der være flere frie

variable, som har det samme navn men *er* forskellige. I et sådant tilfælde vil SCANLOG udskrive **Var_n,** hvor n er et heltal, i stedet for variablens rigtige navn. Dermed undgås udskrivning af samme navn for to forskellige variable.

7.3 Om sproget

SCANLOG sproget er udførligt beskrevet i de øvrige kapitler af bogen. Derfor har vi her kun inkluderet nogle af de vigtigste ting omkring sproget samt nogle mere teknisk betonede ting.

7.3.1 Syntaks

SCANLOG-reglernes syntaks er beskrevet ved eksempler igennem hele bogen. Se især kapitel 4 og 5.

7.3.2 SCANLOG's navne

Variable Alle navne, der begynder med stort er variable. Efter det første bogstav kan der være tal og '_' i navnet (se også afsnit 4.5).

Navne Alle navne, der starter med et lille bogstav, betegner relationer, funktioner eller konstanter (se kapitel 4). Efter det første tegn kan der være tal og '_'.

Strenge Strenge er en sekvens af tegn mellem '. Hvis der skal være en ' inde i en streng, så skal den skrives dobbelt, d.v.s. '', f.eks. 'I' 'm'.
Strenge må maksimalt være på 60 tegn. I kapitel 2 er der mange eksempler med strenge, og i kapitel 5 er der større eksempler, hvor strenge er brugt. Se også afsnit 7.3.5.

Lister En liste starter med '<' og slutter med '>'. En liste kan skilles i hoved og hale med operatoren '..'. Lister er behandlet i kapitel 3.

SCANLOG navne Navnene i figur 7.3 har alle en speciel funktion i SCANLOG og kan derfor kun bruges, som beskrevet for de enkelte. Man kan *ikke* lave sine egne regler for nogen af navnene.

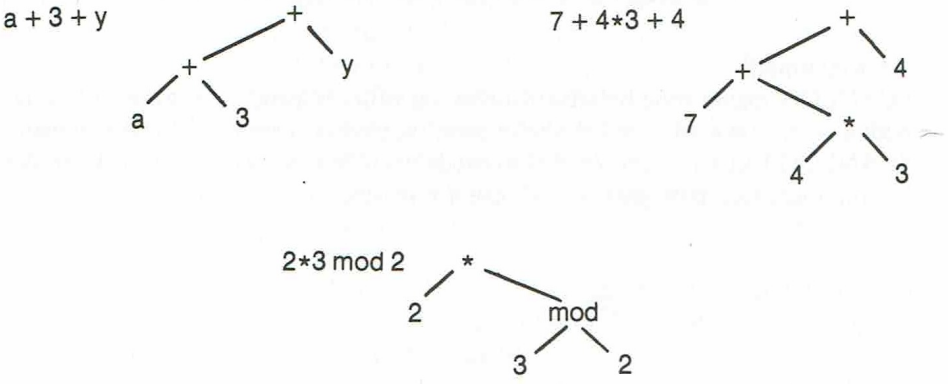
7.3.3 Aritmetik

SCANLOG regner med heltalsaritmetik, og stiller følgende operatorer til rådighed: +, -, *, / og **mod.** / er heltalsdivision, og **mod** er resten ved heltalsdivision. SCANLOG kan kun håndtere heltal mellem -32768 og 32767, så resultater, der kommer udenfor disse grænser, vil ikke nødvendigvis være korrekte.

alle	bekræftes	eksisterer
eller	falsk	find
find_een	for_alle	funktion
glem	gælder	heltal
husk	hvis	hvor
ikke	indsæt	kan
konstant	kopier	liste
længde	læs	mod
nylinie	og	position
sammensæt	sand	skriv
slet	slut	streng
så	variabel	vælg_een
værdi_af		

Figur 7.3 SCANLOG navne.

værdi_af Udregner heltalsværdien af argumentet. Hvis argumentet ikke kan udregnes, f. eks. fordi en variabel ikke har en talværdi, fejler SCANLOG.
Eksempel: X = værdi_af (2+3) binder værdien 5 til X.
værdi_af er nødvendig for at udregne værdien af et udtryk. SCANLOG manipulerer udtryk symbolsk, 2+3 betyder udtrykket 2+3 og *ikke* værdien 5. Det kan man udnytte til f. eks. symbolsk differentiation (se afsnit 5.9), og forenkling af udtryk (se afsnit 5.10). Udtryk er venstreassociative, d. v. s. 2+3+5 opfattes som ((2+3)+5). Det kan man f. eks. se ved at stille spørgsmål som **X+Y = 2+3+5,** hvor **X** bliver **2+3** og **Y** bliver **5.** Ved at stille spørgsmål af denne art, kan man lave en tegning af opbygningen af udtryk (se figur 7.4). Det benyttes blandt andet til forenkling.



Figur 7.4 Tegning af udtryk.

7.3.4 Sammenligningsoperatorer

Tal

Tal kan sammenlignes med disse operatorer:

'=' betyder er lig med
'>' betyder er større end
'<' betyder er mindre end
'=>' betyder større end eller lig med
'<=' betyder mindre end eller lig med
'<>' betyder forskellig fra

Ved '=' kan der blive lavet en variabelbinding.

Strenge

Strenge kan sammenlignes med de samme operatorer som tal, og man ser her på den alfabetiske ordning. Store bogstaver kommer før små. ' ' kommer før alle andre.

'a' > 'A'
'a ' < 'aa'
'abc' > 'ab'

Andre udtryk

Man kan kun bruge '=' og '<>'. To udtryk er ens, hvis deres struktur kan passe sammen. Se nedenstående eksempler (variablene antages at være ubundne).

2 + 3 = Y + X (Y=2 og X=3)
2 * 3 <> Y + X
3 * 2 + 4 = Y + X (Y=3*2 og X=4)
g(a,b) <> g(X)

Der kan ske variabelbindinger ved '='. Et udtryk på formen X <> Y, hvor X og Y kan være hvad som helst, betyder det samme som **ikke (X = Y).**

7.3.5 Strengprimitiver

SCANLOG stiller primitiver til rådighed til manipulation af strenge (se også kapitel 2). I nedenstående angiver de variable, der begynder med S strenge, og variable der starter med X tal. Primitiverne virker alle aftestende, hvis alle parametre er bundet ved anvendelse.

sammensæt (S1,S2,S3)

Primitivet har 3 parametre, og det er sandt, hvis S1 og S2 sammensat giver S3. **sammensæt** kan finde løsninger i flg. tilfælde (blank plads indikerer en ubundet variabel, og S-variablene er bundet til strengene):

sammensæt(S1,S2,)
sammensæt(S1, ,S3)
sammensæt(,S2,S3)
sammensæt(, ,S3)

Primitivet er behandlet i kapitel 2.

længde (S1,X) Er sand, hvis længden af S1 er X. Hvis X ikke er bundet, bliver den sat til længden af strengen.

kopier (S1,X1,X2,S2) Sandt, hvis S1 kopieret fra plads nr. X1 og X2 tegn frem er lig S2. Kan finde løsninger i tilfældet

kopier(S1,X1,X2,)

slet (S1,X1,X2,S2) Sandt, hvis man fra strengen S1 sletter tegnene fra plads nr. X1 og X2 tegn frem og får S2. Kan finde løsninger i nedenstående tilfælde

slet(S1,X1,X2,)

position (S1,S2,X) Primitivet er sandt, hvis positionen af strengen S1 i S2 er X. **position** kan finde løsninger i tilfældet

position(S1,S2,)

7.3.6 Primitiver til læsning og skrivning

Der findes simple primitiver til at skrive og læse fra regler. Se også afsnit 5.13.

skriv Argumenterne skrives ud på skærmen. Hvis variable har fået en værdi, da er denne indsat. Der udskrives ikke linieskift på skærmen. **skriv** kan have et vilkårligt antal argumenter. Ved udskrivning indsættes en blank mellem hvert argument. Strenge udskrives *uden* ', og ' ' i en streng udskrives som '.

nylinie Der udskrives et linieskift på skærmen.

læs Der indlæses fra tastaturet i den variabel, der er argument. Det indlæste *skal* være et normalt SCANLOG argument. Variablen, der indlæses til, skal være fri ved indlæsning, og det indlæste argument bindes til denne variabel.

7.3.7 Kontrolprimitiver

ikke Primitivet er sandt, hvis og kun hvis det der står som argument ikke kan bevises. Der vil ikke blive lavet variabelbindinger. **ikke** kan ikke uddybes.

find_een Primitivet er sandt, hvis argumentet kan bevises. Der findes kun *een* løsning (den første) til argumentet. Variabelbindinger, der laves for at bevise argumentet, vil blive bevaret. Primitivet er brugt i kapitel 2.

eksisterer Som **find_een,** blot bevares variabelbindinger ikke.

vælg_een Kun en af de regler, der er mellem **vælg_een** og **slut** vil blive brugt. En regel kan bruges, hvis hovedet af reglen kan bruges, og en evt. **hvor**-del (se kapitel 4 og afsnit 7.3.8) er sand. Hvis en regel er valgt, men kroppen af reglen ikke kan bevises, vil SCANLOG prøve med de regler, der er efter **slut.** Se også afsnit 4.2.1 og kapitel 2.

for_alle Primitivet kan skrives som vist i nedenstående eksempler

for_alle er_mor_til(X,Y) gælder kvinde(X)
for_alle (f, g og h) gælder (i og j)

Til alle løsninger som findes til spørgsmålet mellem **for_alle** og **gælder,** skal spørgsmålet, der står efter **gælder** (dvs. andet argument), også være sandt. **for_alle** kan ikke uddybes.

7.3.8 Lidt om hvor-delen

I **hvor**-delen (se kapitel 4) af en regel kan man benytte nogle specielle relationer, som SCANLOG stiller til rådighed. Disse er:

variabel(X) Sand hvis X er en fri variabel eller bundet til en fri variabel.
konstant(X) Sand hvis X er bundet til en konstant, f. eks. **joachim.**
heltal(X) Sand hvis X er bundet til et heltal.
funktion(X) Sand hvis X er bundet til en funktion, f. eks. **tid(12,30)** i **møde (tid(12,30),onsdag,jens).**
streng(X) Sand hvis X er bundet til en streng.
liste(X) Sand hvis X er bundet til en liste.

Udtrykkene i **hvor**-delen kan desuden benytte <>, <, >, <=, =>, = og **ikke.** Udtrykkene skrives stort set som kroppen i en regel. Man kan desuden benytte **eller** som vist nedenunder.

```
regnmed(X,Y,Z) hvor
    heltal(X) og X>7 hvis
    .....
regnmed(X,Y,Z) hvor
    (konstant(X) eller heltal(X)) og X <> 7 hvis
    .....
regnmed(X,Y,Z) hvor
    ((heltal(X) og X=7) eller konstant (X)) hvis
    .....
```

Der oprettes ikke variabelbindinger i **hvor**-delen.

7.3.9 Regelmanipulerende primitiver

husk Reglen, der står som argument bliver gemt, som den første med dette navn. Reglen vil stå foran en anden regel med dette navn. Hvis der ikke er andre regler, så vil den nye regel komme til at stå bagest i rækken af regler. Variable, der står i argumentet og som har fået en værdi, vil få værdien indsat i reglen der huskes. Hvis en variabel ikke har fået en værdi, så laver SCANLOG et nyt navn til den. **husk** kan fejle, hvis argumentet ikke er en lovlig regel. Primitivet bevises kun een gang. Primitivet bruges i eksemplet med tilfældige tal side 64.

glem Fjerner den første regel, som kan passe på argumentet til **glem.** Hvis der sker bindinger til variable, så vil disse blive bevaret. **glem** vil ikke blive forsøgt genbevist, hvis noget senere fejler. Primitivet er anvendt side 64.

7.3.10 Kommentarer

Disse kan kun stå *mellem* reglerne (ikke inde i). En kommentar laves ved at skrive et ';' og så skrive kommentaren. Kommentarer kan behandles som andre regler i redigeringssystemet. Der er eksempler på kommentarer i afsnit 5.1.

7.3.11 SCANLOG-disketten

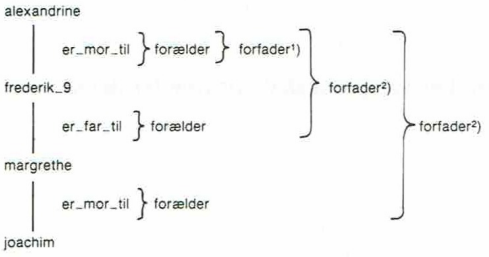
Under kørsel med SCANLOG-systemet *skal* SCANLOG-disketten være i samme diskettestation hele tiden. SCANLOG benytter disketten under redigering og andre ting, og systemet kan ikke fungere uden denne.
Disketten indeholder filerne 'scanlog.cmd', 'scanlog.000', 'scanlog.dat' og 'scanlog.hlp'.

8. Svar til opgaver

De løsninger, der er givet herunder er ikke nødvendigvis de eneste rigtige.

8.1 Svar til opgaver i kapitel 1

1. find alle Barn_1 og Barn_2 så
   søskende(Barn_1,Barn_2).
2. find alle Far så er_far_til(Far,Barn_1),
   er_far_til(Far,Barn_2) og
   Barn_1 <> Barn_2.
3. I 1/ bliver værdien for begge variable udskrevet, i 2/ er det kun værdier for variablen **Far,** der bliver udskrevet. Læg mærke til at der kommer flere ens svar ud på 2/.
   Grunden til at der ikke kommer nogle svar ud i 3/, er at **far** er skrevet med lille i spørgsmålet; dermed tror SCANLOG, at det er navnet på en person, og ikke en variabel.
4. Der er brugt de relationer, som det er vist på fig. 8.1. **Hvordan**-knappen kan bruges til at se, hvilke regler der er brugt.



Figur 8.1 Opgave 4.

5. bedsteforælder(B,Barnebarn) hvis
   forælder(B,Barn) og
   forælder(Barn,Barnebarn).
9. samme_klasse(N1,N2) hvis
   klasse(N1,Årgang,Bogstav,Gren1) og
   klasse(N2,Årgang,Bogstav,Gren2).
10. find Fag så
    klasse(siigurd_F,Årgang,Bogstav,Gren) og
    timeplan(Årgang,Bogstav,Fag,X,9).
11. find alle Lærer så
    klasse(siigurd_F,Årgang,Bogstav,Gren),
    timeplan(Årgang,Bogstav,Fag,Dag,9),
    lærerplan(Årgang,Bogstav,Fag,Lærer).

12. lærere(N1,Lærer,Klokken) hvis
    klasse(N1,Årgang,Bogstav,Gren) og
    timeplan(Årgang,Bogstav,Fag,Dag,Klokken).
    find alle Lærer så lærere(N1,Lærer,X1) og
    lærere(N2,Lærer,X2).
13. Man har timer sammen med en anden klasse, hvis man på det samme tidspunkt har den samme lærer.

    find alle Årgang1 og Bogstav1 så
    klasse(siigurd_F,Årgang,Bogstav,Gren),
    timeplan(Årgang,Bogstav,Fag,Tid),
    timeplan(Årgang1,Bogstav1,Fag,Tid),
    Årgang <> Årgang1,
    Bogstav <> Bogstav1,
    lærerplan(Årgang,Bogstav,Fag,Lærer) og
    lærerplan(Årgang1,Bogstav1,Fag,Lærer).

8.2 Svar til opgaver i kapitel 2

1. Et svar er 'no I like me'.
2. Du tilføjer

    lav_om('You', 'I').

    o.s.v. indeni **vælg_een**-konstruktionen. De nye fakta skal komme *før* det sidste faktum (**lav_om(Ukendt,Ukendt)**).
4. venstre(Streng,Tegn,Rest) hvis
    kopier(Streng,1,1,Tegn) og
    sammensæt(Tegn,Rest,Streng).
    højre(Streng,Tegn,Rest) hvis
    længde(Streng,Længde),
    kopier(Streng,Længde,1,Tegn) og
    sammensæt(Rest,Tegn,Streng).
5. venstre_bogstav(Streng,Bogstav,Rest) hvis
    venstre(Streng,Bogstav,Rest) og
    Bogstav <> ' '.
    venstre_bogstav(Streng,Bogstav,Rest) hvis
    venstre(Streng,B,R),
    B = ' ' og
    venstre_bogstav(R,Bogstav,Rest).
    højre_bogstav(Streng,Bogstav,Rest) hvis
    højre(Streng,Bogstav,Rest) og
    Bogstav <> ' '.
    højre_bogstav(Streng,Bogstav,Rest) hvis
    højre(Streng,B,R),
    B = ' ' og
    højre_bogstav(R,Bogstav,Rest).

6. palindrom(' ').
   palindrom(Streng) hvis
       venstre_bogstav(Streng,Bogstav,Rest1),
       højre_bogstav(Rest1,Bogstav,Rest2) og
       palindrom(Rest2).

8.3 Svar til opgaver i kapitel 3

1. har_børn(margrethe,<frederik,joachim>).
2. er_mor_til(Mor,Barn) hvis
       har_børn(Mor,Børn) og
       er_med_i(Barn,Børn).
3. find_hale(<Hoved..Hale>,Hale).
4. tilføj_een(Element,Liste,<Element..Liste>).
5. plads(1,<Element..Hale>,Element).
   plads(X,<Hoved..Hale>,Element) hvis
       X>1 og
       plads(værdi_af(X-1),Hale,Element).
6. naboer(X,Y,<X,Y..Hale>).
   naboer(X,Y,<Hoved..Hale>) hvis
       naboer(X,Y,Hale).
7. reverser(<>,<>).
   reverser(<El..Hale1>,Resultat) hvis
       reverser(Hale1,Res1) og
       konkatener(Res1,<El>,Resultat).
   eller
   reverser(L1,L2) hvis
       revkat(L1,<>,L2).
   revkat(<E..Hale>,L1,L2) hvis
       revkat(Hale,<E..L1>,L2).
   revkat(<>,L,L).
8. sidste(E,<E>).
   sidste(E,<Hoved..Hale>) hvis
       sidste(E,Hale).
9. substituer(E1,E2,<>,<>).
   substituer(E1,E2,<E1..Hale1>,<E2..Hale2>) hvis
       substituer(E1,E2,Hale1,Hale2).
   substituer(E1,E2,<Hoved..Hale1>,<Hoved..Hale2>) hvis
       E1<>Hoved og
       substituer(E1,E2,Hale1,Hale2).
10. er_med_i(E,Liste) hvis
       konkatener(L,<E..Hale>,Liste).
    sidste(E,Liste) hvis
       konkatener(L,<E>,Liste).
    naboer(X,Y,Liste) hvis
       konkatener(L,<X,Y..Hale>,Liste).

11. delmængde(<>,L).
    delmængde(<Hoved..Hale>,L) hvis
        er_med_i(Hoved,L) og
        delmængde(Hale,L).
12. fællesmængde(<>,X,<>).
    fællesmængde(<X..R>,Y,<X..Z>) hvis
        er_med_i(X,Y) og
        fællesmængde(R,Y,Z).
    fællesmængde(<X..R>,Y,Z) hvis
        ikke(er_med_i(X,Y)) og
        fællesmængde(R,Y,Z).
    foreningsmængde(<>,X,X).
    foreningsmængde(<X..R>,Y,Z) hvis
        er_med_i(X,Y) og
        foreningsmængde(R,Y,Z).
    foreningsmængde(<X..R>,Y,<X..Z>) hvis
        ikke(er_med_i(X,Y)) og
        foreningsmængde(R,Y,Z).
    krydsprodukt(<>,X,<>).
    krydsprodukt(<El..Hale>,Liste,Resultat) hvis
        produkt(<El,Liste,Res1),
        krydsprodukt(Hale,Liste,Res2) og
        konkatener(Res1,Res2,Resultat).
    produkt(E,<>,<>).
    produkt(El,<E2..Hale1>,<<El,E2>..Hale2>) hvis
        produkt(El,Hale1,Hale2).

8.4 Svar til opgaver i afsnit 5.2

1. find Resultat så bøj('parler','il',Resultat).
2. Et svar er 'nous' og 'finesser'
   Et svar er 'nous' og 'finir'
   Et svar er 'je' og 'finissonre'
   Et svar er 'tu' og 'finissonre'
   Et svar er 'nous' og 'finissre'
   Ikke flere løsninger
3. Être hedder i første person flertal: sommes, men spørger man SCANLOG

   find Resultat så bøj('etre','nous',Resultat)

   så fås svaret

   Et svar er 'etons'

4. Reglerne er vist på filen **uverber.** Husk at fjerne de eksisterende regler, før den nye fil læses ind.

5. Tilføj regler som

```
bøjning(imparfait,1,'je','ais').
bøjning(imparfait,1,'tu','ais').
bøjning(imparfait,1,'il','ait').
.
.
```

6. Det er en god ide at tage de verber, som kun har få uregelmæssigheder. Læg mærke til at der kommer flere løsninger ud, når SCANLOG spørges

```
find alle Resultat så
    bøj(present,'boire','nous',Resultat).
```

8. I vælg_een konstruktionen med **bøj** kan der indsættes en regel, som denne

```
bøj(passe_compose,Verbum,Person,Resultat) hvis
    bøj(present,'avoir',Person,R1),
    find_type(Verbum,Type,Stamme),
    sammensæt(R1,' ',R2),
    sammensæt(R2,Stamme,R3) og
    sammensæt(R3,'e',Resultat)
```

8.5 Svar til opgaver i afsnit 5.4

1. Spørg SCANLOG,

```
find Resultat så kod('fritime',Resultat).
```

2. koldt øl
4. Koden ville ikke mere være entydig, for strengen '.- ' kunne nu både være tegnet 'a' og teksten 'et '.

8.6 Svar til opgaver i afsnit 5.6

1. Tæl det antal løsninger, der fremkommer på spørgsmålet

```
find alle X så
    syntese(X,<lysin,valin,cystein,valin>).
```

2. I stedet for at spørge om hvilke aminosyrer.

```
a c g c c a a c g t t c
```

giver, kan du f. eks. spørge om rækkefølgen

```
a c g c a a c g t t c c
```

3. Mange steder vil det ikke nødvendigvis give en forskel, men i mange tilfælde vil en enkelt aminosyre blive ændret.
4. Koderne er næsten ens, for da vil en enkelt mutation med lidt held give den oprindelige aminosyre (og dermed det oprindelige protein) igen.
5. Hvis der kommer en eneste mutation, vil syntesen gå i stå, og det er ikke sikkert, at en enkelt mutation er fatal. En biologilærer vil yderligere kunne uddybe dette svar.

8.7 Svar til opgaver i afsnit 5.15

1. **int**-reglerne laver heltal fra 0 og opefter.

2. Den anden og tredje regel med **test_gæt** skal ændres, så de bliver

```
test_gæt(Kode,Gæt,Gange) hvor
    heltal(Gæt) og
    Kode <Gæt hvis
    skriv('For højt'),
    nylinie,
    skriv('Et tal mellem 0 og 99 (eller stop) ? '),
    læs(Ny_Gæt) og
    test_gæt(Kode,Ny_Gæt,værdi_af(Gange+1)).
```

   Den tredje regel bliver som før.

9. Litteratur

Den interesserede læser kan søge mere information om logikprogrammering i de nedenfor angivne bøger. Vi har forsøgt, at anbringe dem i en rækkefølge, så de først angivne er de lettest tilgængelige. En del af litteraturen er på engelsk; der findes ikke dansk litteratur i større udstrækning.

Bøgh, Sestoft og Skardhamar:
5-timers Introduktionskursus i micro-PROLOG
Prolog-Data-DK 1984.

Jonathan Briggs:
micro-PROLOG, fakts og regler
Prolog-Data 1985.

Richard Ennals:
Beginning micro-PROLOG
Ellis Horwood 1983.

Hakun F. Skardhamar:
Syntaksanalyse (parsing)
Prolog-Data-DK 1984.

W. F. Clocksin og C. S. Mellish:
Programming in Prolog
Springer Verlag 1981.

K. L. Clark og F. G. McCabe:
micro-PROLOG: Programming in Logic
Prentice-Hall International 1984.

J. A. Campbell:
Implementations of PROLOG
Ellis Horwood 1984.

Indeks

SCANLOG

– et DANSK LOGIK-PROGRAMMERINGS-SPROG af Kurt Fleckner & Jan Rubæk Pedersen

Logikprogrammering er grundlaget for fremtidens vidensbaserede systemer, som også tænkes anvendt i japanernes nye såkaldte »femte-generations«-computere. Derfor er der væsentlige argumenter for at begynde at beskæftige sig med logikprogrammering.

SCANLOG er det første dansk udviklede logik-programmeringssprog til undervisningsområdet (gymnasiet, HF, folkeskolens ældste klassetrin og seminariet).

SCANLOG er et dialog-orienteret femte-generations-sprog, der frigør brugeren for den detaljerede del af programmeringsarbejdet.

SCANLOG-systemet udmærker sig ved at være et brugervenligt programmeringssprog med rette- og redigeringsfaciliteter på dansk. Systemet er menudrevet ved hjælp af maskinens funktionstaster. Ved fejlmeldinger gives oplysende hjælpetekster. En »hvordan-knap« forklarer, hvordan SCANLOG »tænker«.

I tilknytning til bogen er udarbejdet en diskette med mange undervisningseksempler, øvelser og løsningsforslag fra en bred fagrække (f.eks. franske verber, DNA-kodning, symbolsk differentiation, reduktion af matematiske udtryk, stamtræ, sortering samt forskellige problemløsningsopgaver og -strategier).

SCANLOG kan benyttes på Partner/Piccoline i såvel grundfiguration som net samt på IBM PC.

FORLAGET SFU

ISBN 87-7260-005-5