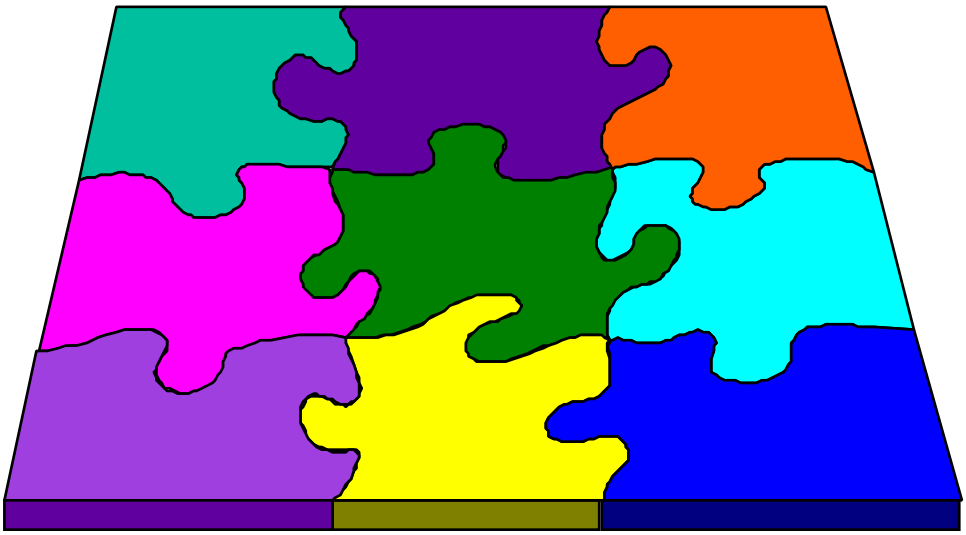
Forslag til datalogiprojekter

Keld Helsgaun
RUC, januar 1998



I det følgende giver jeg en række forslag til projekter på overbygningsuddannelsen i datalogi.

Enkelte af forslagene er blot angivet på overskriftsform, men jeg har i disse tilfælde forsøgt at gøre overskrifterne så sigende som muligt og har anført litteratur, som kan være relevant i projektopstarten. Overskrifterne skulle forhåbentlig give en ide om projekternes indhold, men for en nærmere uddybning er I meget velkomne til at kontakte mig. Dette gælder naturligvis også, hvis I selv har forslag eller ideer til projekter.

Ud for hver projektoverskrift er anført numrene på de moduler, som projektet vil kunne henføres til.

Indholdsfortegnelse

1. Planters algoritmiske skønhed (modul 1, 2 og 3)

Planters former har optaget matematikere i århundreder. Planters smukke geometriske træk, såsom symmetrien i blade og blomster, har været genstand for intensive studier. Målet har været at udtrykke planters *udseende* på en matematisk form.

I 1968 introducerede en biolog, A. Lindenmayer, en ny matematisk teori for planters *udvikling* [1]. Han indførte de såkaldte Lindenmayer-systemer (L-systemer) til at formulere algoritmer for, hvordan planter udvikler sig.

L-systemer giver en elegant notation til modellering og simulering af planters udvikling. Tidsmæssige forløb udtrykkes ved hjælp af relativt simple grammatiske regler. Et stadium i en plantes udvikling beskrives ved hjælp af en tegnfølge, og de grammatiske regler udtrykker, hvorledes tegnfølgen kan omskrives, dvs. ændres, så den repræsenterer et efterfølgende stadium i plantens udvikling.

Et L-system kan gives en grafisk tolkning gennem såkaldt *skildpaddefortolkning* af de indgående tegnfølger. Hvert tegn i følgen fortolkes som en kommando til en skildpadde, der kan bevæge sig på et stykke papir i forskellige retninger og tegne undervejs. Skildpaddens tilstand er defineret ved triplet (x,y,α), hvor (x,y) er koordinaterne for dens position, og α er en vinkel, der angiver skildpaddens orientering.

Givet en skridtlængde d og en vinkeltilvækst δ, så kunne kommandoerne til skildpadden for eksempel være følgende:

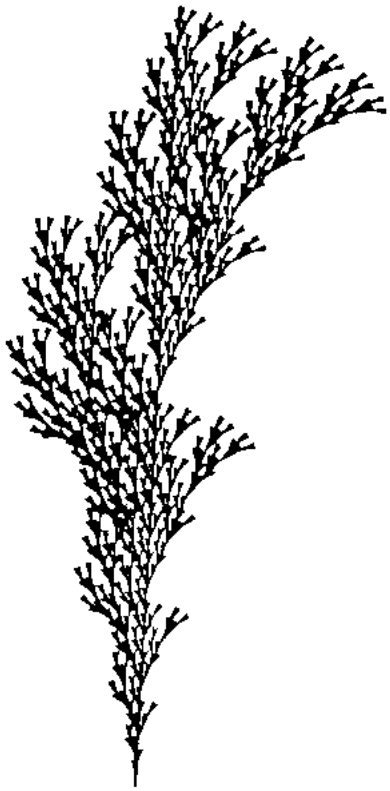
F	Gå et skridt af længde d fremad og tegn samtidig.
f	Gå et skridt af længde d fremad uden at tegne.
+	Drej vinklen δ til venstre.
-	Drej vinklen δ til højre.
[	Gem den aktuelle tilstand på en stak.
]	Lad den aktuelle tilstand være den øverste på stakken og afstak.

Jeg vil her nøjes med at give et enkelt eksempel, der illustrerer slagkraften af L-systemer.

```
n=4,  d=22°
F
F     F[+F]F[-F][F]
```

I første linje angives, at antallet af omskrivninger skal være 4, mens vinkeltilvæksten skal være 22°. Anden linje udtrykker, at starttegnfølgen skal være F. I den tredje linje defineres en omskrivningsregel, som udtrykker, at F overalt i en tegnfølge kan erstattes med udtrykket på højresiden af pilen.

Ved at fortolke den resulterende tegnfølge ved brug af skildpaddegrafik fås følgende busklignende vækst.



Det nævne eksempel er taget fra Prisinkiewicz og Lindendmayers imponerende bog [2]. Bogen giver en grundig indføring i L-systemer og er rigt illustreret med grafiske udtegninger, heriblandt mange i farver.

Jeg foreslår et datalogisk projekt, der med udgangspunkt i denne bog, har som mål at få udviklet et system til eksperimentering med L-systemer. Systemet skal kunne indlæse et L-system, simulere udviklingen og derefter vise resultatet grafisk ved hjælp af skildpaddefortolkning.

[1] A. Lindenmayer,
Mathematical models of cellulare interaction in development. Part I and II,
Journal of Theoretical Biology. 18:280-315, 1968.

[2] P. Prusienkiewicz and A. Lindenmayer,
The Algorithmic Beauty of Plants,
Springer Verlag, 1990.

2. Systemsimulering (modul 1, 2 og 3)

Her er ikke tale om et specifikt projektforslag, men snarere en temaoverskrift. Indenfor temaet kan f.eks. laves projekter af typen:

- simulering af X (hvor X er et eller andet system)
- undersøgelse af simuleringsparadigmer
- simuleringssprog
- effektiv organisering af hændelseskøer
- animation

Ved simulering forstås en efterligning af et tidsligt forløb.

Som eksempler på systemer, hvor simulering med fordel er blevet anvendt, kan nævnes:

- Menneskets kredsløb
- Solsystemets dannelse
- Økosystemer (f.eks. livet i en sø)
- Kødannelse (f.eks. lægekonsultationer og posthuse)
- Trafik (f.eks. S-togstrafik)
- Procesanlæg (f.eks. kraftværker)
- Opsendelse af rumraketter
- Legemers bevægelse (f.eks. himmellegemer eller billardkugler)

Disse eksempler er blot tænkt som inspiration til mulige projektvalg. Det skal imidlertid understreges, at der allerede i projektets start bør foreligge en model af det system, der ønskes simuleret. Desuden bør inddata være tilgængelige. Hvis disse to forudsætninger ikke er opfyldt, er det sædvanligvis umuligt, inden for projektets afgrænsede tidsramme, at nå frem til selve simuleringen af systemet. Al tiden vil nemlig så gå med opstilling af model, dataindsamling og verifikation af modellen. Det kan i denne forbindelse nævnes, at jeg har mulighed for at fremskaffe modeller og inddata til de fleste af ovennævnte eksempler.

Sædvanligvis skelner man skarpt mellem to typer af simulering, nemlig *kontinuert* simulering og *diskret* simulering.

I en kontinuert simulering er den dynamiske model udtrykt ved en sæt af koblede differentialligninger. Simuleringen omfatter numerisk løsning af de indgående differentialligninger.

I en diskret simulering er modellen udtrykt ved hjælp af *hændelser*, d.v.s. øjeblikkelige ændringer af systemets tilstand. Metoder til fremskrivning af modeltiden til næste hændelse er centrale i denne type af simuleringer.

Herudover kan det i visse situationer være hensigtsmæssigt at sammenblande de to simuleringstyper, nemlig i form af *kombineret kontinuert og diskret simulering*. Som et simpelt eksempel kan nævnes simulering af et køleskab. Den kontinuerte ændring af køleskabets temperatur som følge af dets varmeudveksling med omgivelserne kan udtrykkes ved hjælp af en sædvanlig førsteordens differentialligning. Termostaten, som slår til og fra, når temperaturen i køleskabet passerer visse tærskelværdier, giver anledning til hændelser. Et andet eksempel er simulering af blodsukkerkoncentrationen hos sukkersyge. Den kontinuerte variation af blodsukkerkoncentrationen ændres pludseligt som følge af fødeindtagelse og insulininjektioner.

Der findes mange udmærkede indføringer i simuleringsteknik, bl.a. [1]. En udmærket indføring i simuleringsparadigmer findes i [2].

[1] R. E. Shannon,
Systems simulation: the art and science,
Prentice-Hall, 1975.

[2] W. Kreutzer,
System simulation: programming styles and languages.,
Addison-Wesley, 1986.

[3] J. G. Vaucher,
Comparison of simulation event list algorithms.,
Comm. ACM, Vol. 18, 1975 (pp. 223-230).

[4] J. H. Kingston,
Analysis of Three Algorithms for the Simulation Event List,
Acta Informatica, Vol. 22, April 1985 (pp. 15-34).

3. Diskret simulering i Java (modul 1 og 2)

Java er et relativt nyt programmeringssprog. Sproget tillader programmøren at skrive platformuafhængig kode for såvel konventionelle som internetbaserede anvendelser.

Java er objektorienteret og har i denne henseende mange lighedspunkter med Simula. Java og Simula adskiller sig imidlertid fra hinanden på flere punkter. En væsentlig forskel er de to sprogs muligheder for at udtrykke parallelitet. Simula tillader brugen af pseudo-parallelitet (ved hjælp af korutiner), hvorimod Java tillader ægte parallelitet (ved hjælp af tråde).

Det er velkendt, at diskret simulering kan realiseres ved hjælp af korutiner, altså ved brug af pseudo-parallelitet. Simula indeholder således et værktøj til diskret simulering, klassen Simulation, hvis implementering er baseret på Simulas korutinebegreb. Klassen kan programmeres i Simula ved brug af korutineprimitiverne Resume og Detach (se f.eks. [1]).

Java indeholder ikke tilsvarende korutineprimitiver. Spørgsmålet er derfor, om det i Java kan lade sig gøre at implementere et værktøj til diskret simulering, der svarer til Simula-klassen Simulation. Projektet skal forsøge at besvare dette spørgsmål.

[1] H. B.. Hansen,
Simula - et objektorienteret programmeringssprog,
RUC, 1992.

[2] H. M. Deitel and P. J. Deitel.
Java. How to program.
Prentice Hall, 1996.

[3] G. Bruno,
Using Ada for Discrete Event Simulation,
Software - Practice and Experience, Vol. 14, No. 7, 1984 (pp. 685-695).

[4] R. M. Bryant,
Discrete system simulation in Ada.
SIMULATION, Vol. 14, No.39, 1982 (pp. 111-121).

4. Genetiske algoritmer (modul 1, 2 og 3)

I 1839 udgav Charles Darwin sit hovedværk, *The Origon of the Species*, hvori han fremlagde princippet om evolution gennem naturlig udvælgelse:

- Hvert individ vil videreføre egenskaber til sit afkom.
- Ikke desto mindre producerer naturen forskellige individer.
- De stærkeste individer får mere afkom end de svage, hvorved populationen som helhed får fordelagtige egenskaber.
- Over en lang periode kan variation ophobes og resultere i nye arter med særlige egenskaber.

Darwins evolutionsprincip er i dag almindeligt accepteret, men ikke mange ved, at princippet kan bruges til at konstruere effektive optimeringsalgoritmer, de såkaldte *genetiske algoritmer*. Denne type af algoritmer blev introduceret af J. H. Holland i 1986 [1].

En genetisk algoritme simulerer udviklingen i en population. Hvert individs arveanlæg er fastlagt ved dets gener. Individerne kan parre sig, hvorved afkommet arver nogle af forældrenes egenskaber efter fastlagte regler. Et "stærkt" afkom er et individ med egenskaber, som svarer til en løsning tæt på optimum.

Idet sandsynligheden for overlevelse stiger med individets styrke, vil populationen tendere mod stærke individer. Efter et vist antal generationer standses algoritmen med en population, hvor det stærkeste af individerne svarer til en løsning tæt på optimum.

Projektet går ud på at afprøve genetiske algoritmers effektivitet gennem et, eventuelt flere, udvalgte eksempelproblemer. Et muligt eksempel kunne være *den rejsende sælgers problem*, som, kort fortalt, går ud på at finde den korteste rejserute for en sælger, der skal besøge en række byer. Men der er også andre muligheder. En god ide kunne være, i første omgang, at forsøge at løse det problem, som P. H. Winston benytter i sin lærebog om kunstig intelligens [2], nemlig oplæring af en bager til at optimere mængden af sukker og mel i sine kager.

[1] J. H. Holland, K. J. Holyoak, R. E. Niebett and P. R. Thagard,
Induction: Processes of Inference, Learning and Discovery,
MIT Press, 1986.

[2] P. H. Winston,
Artificial Intelligence,
Addison-Wesley, 1992 (3rd ed.).

5. Kombinatorisk optimering (modul 1, 2 og 3)

Området *kombinatorisk optimering* omhandler løsning af problemer, hvor der blandt et endeligt, men ofte meget stort, antal muligheder skal bestemme en optimal løsning. Antallet af muligheder kan i visse tilfælde antage astronomiske størrelser, f.eks. 10^{10000}, hvilket nødvendiggør, at særdeles effektive søgemetoder må tages i anvendelse.

Blandt søgemetoderne skelnes der mellem *eksakte* metoder og *approksimative* metoder. Med de eksakte metoder bestemmes det eksakte optimum for et givet problem. De approksimative metoder giver derimod kun en tilnærmelse til optimum, men med et relativt lille tidsforbrug.

En eksempel på en approksimativ metode er *simuleret nedkøling* (engelsk: simulated annealing) [1]. Metoden, der er baseret på en fysisk analogi, langsom nedkøling, har vist sig effektiv i forbindelse med løsning af mange kombinatoriske optimeringsproblemer. Dens styrke ligger i dens evne til at undslippe lokale optima.

Et andet eksempel er *tabusøgning* [2][3]. Denne metode, der er forholdsvis ny, har i flere tilfælde vist sig at være mere effektiv end simuleret udglødning.

Jeg kunne forestille mig et projekt, hvor en af de to metoder (eller dem begge), studeres og afprøves på et simpelt eksempel. Som eksempel kunne f.eks. vælges farvelægningsproblemet for grafer [4].

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi,
Optimization by simulated annealing,
Science, Vol. 220, 1983 (4458).

[2] F. Glover,
Future paths for integer programming and links to artificial intelligence,
Computers Oprns. Res., Vol. 13, 1986 (pp. 533-549).

[3] F. Glover,
Tabu Search - Part I,
ORSA J. Comput., Vol. 1, 1989 (pp. 190-206).

[4] A. Hertz and D. de Werra,
Using Tabu Search Techniques for Graph Coloring,
Computing, Vol. 39, 1987 (pp. 344-351).

6. Myrekolonier til kombinatorisk optimering (modul 1 og 3)

Jo, du læste faktisk rigtigt. Det er muligt at løse visse problemer ved at efterligne dyrs evner til at løse problemer. I dette tilfælde efterlignes myrers instinktive evne til at finde vej i et ukendt terræn. Metoden er inspireret af forskning i myrers kollektive adfærd. Etologerne har forsøgt at forstå, hvordan de næsten blinde myrer kan finde den korteste vej fra myretuen hen til et sted med føde. En hypotese er, at myrerne 'kommunikerer' ved hjælp af deres ekskrementer. En myre, der bevæger sig, lægger undervejs sine ekskrementer, og når en anden myre så senere kommer til et sted på denne vej, vælger den med høj sandsynlighed at følge den forrige myres spor. Derved forstærkes 'lysten' hos andre myrer til at følge sporet.

Der er med andre ord tale om kollektiv indlæring af en hensigtsmæssig adfærd. Datalogisk udtrykt, er der tale om "distribuerede beregninger foretaget af samarbejdende, men ikke centralt kontrollerede, processorer". Problemløsningsmetoden er naturligvis særlig relevant i forbindelse med løsning af problemer på paralleldatamater, men også på traditionelle datamater har metoden vist sin berettigelse.

I dette projekt implementeres og afprøves en algoritme, inspireret af myrekoloniers adfærd, til løsning af et klassisk kombinatorisk optimeringsproblem, nemlig "Den rejsende sælgers problem". Problemet går i korthed ud på at bestemme den korteste rejserute for en person, der skal besøge en række byer.

Som udgangspunkt for projektet benyttes nedenstående artikler, specielt den første af referencerne.

[1] M. Dorigo and L. M. Gambardella,
ANT-Q. A Cooperative Learning Approach to Combinatorial Optimization,
Technical Report 95-01. IRIDIA, Université Libre de Bruxelles, 1995.

[2] A.. Colorni, M. Dorigoand V. Maniezzo,
Distributed Optimization by Ant Colonies,
Proc. ECAL91, Paris 1991, pp. 134-142.

[3] A.. Colorni, M. Dorigo and V. Maniezzo,
An Investigation of some properties of an "Ant algorithm",
Proc. PPSN,, Bussels, 1992 (pp. 509-520).

[4] M. Dorigo, V. Maniezzo and A.. Colorni,
The Ant System: Optimization by a colony of cooperating agents,
IEEE Trans. Sys., Man and Cyb.-part B. Vol. 26, No. 1, 1996 (pp. 1-13).

7. **Kryptografering** (modul 1)

Kryptografering betegner metoder til at kode meddelelser, så de bliver svære at tyde for uvedkommende, som måtte opfange meddelelserne. Kryptografering anvendes for eksempel, når 'følsomme' data skal overføres i et datanetværk. For eksempel ved overførsel af PIN-koder fra Dankort-automater.

Kryptografering omfatter to processer, kaldet henholdsvis kryptering og dekryptering. Ved en krypeteringen foretages en omformes meddelelsen, en tekst T, til en kodet tekst, C, ved hjælp af krypteringsfunktionen f:

$$C = f(T)$$

Dekryptering er den modsatte operation, som foretages ved hjælp af den inverse funktion, f^{-1}:

$$T = f^{-1}(C)$$

Krypteringsfunktionen f defineres ofte i form af en algoritme. I dette projekt skal den såkaldte RSA kryptograferingsmetode implementeres. Ved denne metode foretages både kryptering og dekryptering ved "potensopløftning":

$$C = T^{k} \bmod N$$
$$T = C^{k'} \bmod N$$

For at virke efter hensigten (være svær at afkode for uvedkommende) bør k, k' og N vælges med omhu. Et rimeligt godt valg for N er en værdi, som er produktet af to store primtal. Der er således behov for at kunne foretage beregninger på heltal med vilkårligt mange cifre. Derfor foreslås det, at der som grundlag for implementeringen af RSA udvikles en generel Java-klasse til brug for denne type aritmetiske beregninger (multipel-præcisions-aritmetic). Som udgangspunkt kan programmet i den første af nedenstående referencer benyttes (s.343.356). I forhold til dette program bør bl.a. multiplikationsalgoritmen effektiviseres, som beskrevet på siderne 357-361 i referencen, eller måske endnu bedre, som beskrevet i den anden af de to referencer.

[1] H. Riesel,
Prime Numbers and Computer Methods for Factorization,
Birkhäuser, 1994.

[2] *Numerical Recipes in C: The Art of Scientific Computing*,
Chap. 20.6: Arithmetic of Arbitrary Precision (pp. 915-925).

8. Parallelle algoritmer (modul 1, 2 og 3)

Overskriften skal blot ses som et tema. To mulige projekter kunne være at undersøge (1) parallelle algoritmer til sortering og (2) parallelle algoritmer til løsning af grafproblemer.

[1] H. T. Kung,
The Structure of Parallel Algorithms,
Advances in Computers (ed. M. C Yovits), Vol. 19, 1980 (pp. 65-112).

[2] D. J. Kuck.,
A Survey of parallel machine organization and programming.
Computing Surveys, Vol. 9, No. 1, 1977 (pp. 29-59).

[3] S. Lakshmivarahan, S. K. Dhall and L. L. Miller,
Parallel Sorting Algorithms,
Advances in Computers (ed. M. C. Yovits), Vol. 23, 1984 (pp. 295-394).

[4] K. M. Chandy and J. Mishra,
Distributed Computations on Graphs: Shortest Path Algorithms,
Comm. ACM, Vol. 25, No. 11, 1982 (pp. 833-837).

[5] F. Y. Chin, J. Lam and I. Chen,
Efficient Parallel Algorithms for Some Graph Problems,
Comm. ACM, Vol. 25, No. 9, 1982 (pp. 659-665).

[6] J. Mishra and K. M. Chandy,
A Distributed Graph Algorithm: Knot Detection,
ACM. Trans. Prog. Lang. and Syst., October 1982.

[7] S. G. Akl,
Parallel Sorting Algorithms,
Academic Press, 1985.

9. NESL - et sprog til parallelprogrammering (modul 2)

NESL er et nyt sprog til parallelprogrammering, dvs. udvikling af programmer, som tillader samtidig udførelse af operationer på data. Sproget er forholdsvist simpelt og er velegnet både til undervisning i parallelprogrammering og til udvikling af parallelle algoritmer.

En grundlæggende egenskab ved NESL er, at sproget tillader samtidige operationer på sekvenser - endimensionale tabeller. For at give et indtryk af nogle af sprogets muligheder er nedenfor vist en implementering i NESL af Quicksort, en velkendt algoritme til sortering af tabeller.

```
function Quicksort(S) =
if (#S <= 1) then S
else
  let a = S[rand(#S)];
    S1 = {e in S | e < a};
    S2 = {e in S | e == a};
    S3 = {e in S | e > a};
    R = {Quicksort(v): v in [S1,S3]};
  in R[0] ++ S2 ++ R[1] $
```

Operatoren `#` returnerer længden af en sekvens. Funktionen `rand(n)` returnerer et tilfældigt heltal mellem `0` og `n`. Udtrykket `S[rand(#S)]` returnerer således et tilfældigt element i `S`. Notationen `{e in S | e < a}` betyder: "find i parallel alle elementer `e` i `S`, for hvilke `e < a`". Notationen `{Quicksort(v): v in [S1,S3]}` betyder "udfør `Quicksort(v)` i parallel for alle `v` i `S1` og `S3`". Operatoren `++` sætter to sekvenser i forlængelse af hinanden.

Oversætteren til NESL oversætter til et mellemsprog, der kaldes VCODE. Den resulterende mellemkode kan så fortolkes, eller oversættes til en paralleldatamat, f.eks. til Connection Machines CM-2/CM-5 eller Cray C90.

Der er mange muligheder for projekter, der omhandler NESL. Blandt disse kan nævnes:

(1) Udvikling af en NESL-oversætter.
(2) Udvikling af en VCODE-fortolker.
(3) Evaluering af NESL.

[1] G. E. Blelloch,
Programming Parallel Algorithms,
C. ACM, Vol. 39, No. 3, 1996 (pp. 85-97).

10. Rettelse af stavefejl (modul 1)

Implementering og afprøvning af en algoritme til afsløring og rettelse af stavefejl.

[1] J. L. Peterson,
Computer programs for detecting an correcting spelling errors,
Comm. ACM, Vol. 23, No. 12, 1980 (pp. 676-687).

[2] M. Mor, A. S. Fraenkel,
A Hash Code Method for Detecting and Correcting Spelling Errors,
Comm. ACM, Vol. 25, No. 12, 1982 (pp. 935-938).

11. Intervalaritmetik (modul 1)

Det er velkendt, at data kan være forbundet med usikkerhed. Tænk for eksempel på data, som aflæses på måleinstrumenter. Alligevel foretages der ofte beregninger på sådanne data, som om der var tale om eksakte data. Ofte undlades egentlige følsomhedsanalyser, hvor den beregningsmæssige betydning af dataenes usikkerhed analyseres. Man stoler på, at de beregnede resultater kan bruges, på trods af inddataene er fejlbehæftede.

Dette projekt går ud på at stille et værktøj til rådighed, som kan benyttes i forbindelse med følsomhedsanalyser, nemlig en Java-klasse, der kan regne med intervaller. Programmellet skal kunne foretage aritmetiske beregninger på tal, hvis fejlen kan angives i form af intervaller.

Hvis værdien for x vides at ligge i intervallet [xmin,xmax], så skal alle beregninger, hvor x indgår benytte dette interval. Antag f.eks. at x tilhører intervallet [xmin,xmax] og y tilhører intervallet [ymin,ymax], så kan det konkluderes, at deres sum x+y tilhører intervallet [xmin+xmax,ymin+ymax]. Tilsvarende intervalregneregler kan angives for andre aritmetiske operationer.

For at tilbyde en bekvem brugergrænseflade anbefales et objektorienteret design. Et tals interval er et objekt med tilhørende aritmetiske operationer. Således kan addition af x og y (hvor x og y er talintervaller) f.eks. udtrykkes ved x.Add(y).

Alternativt kunne projektet rette sig mod C^{++} i stedet for Java.

12. En Java-pakke til tekstbehandling (modul 1)

Design og implementering af en generel Java-pakke, der indeholder de grundlæggende byggesten til konstruktion af tekstbehandlingsprogrammer (f.eks. tekstredigeringsprogrammer). Ideer til faciliteter, som klassen skal indeholde kan f.eks. hentes i sproget SNOBOL [1]. Alternativt kunne projektet rette sig mod C^{++} i stedet for Java.

[1] K. E. Griswold, J. F. Poage and I. P. Polonsky,
The SNOBOL 4 Programming Language,
Prentice Hall, 1971.

[2] K. C. Liu,
A String Pattern Matching Extension to Pascal,
Software - Practice and Experience, Vol. 16, No. 6, 1986 (pp. 541-548).

13. Robotstyring (modul 1 og 2)

Et sprog til styring af robotter.

[1] C. Blume and W. Jacob,
PasRo. Pascal and C for Robots.,
Springer Verlag, 1985.

[1] C. Blume and W. Jacob.,
Programming Languages for industrial Robots,
Springer Verlag, 1985.

14. Problemløsning med båndlagte variable (modul 1 og 3)

I nedennævnte artikel af Sussmann og Steele [1] beskrives et sprog, CONSTRAINTS, til løsning af problemer, der involverer variable, som er båndlagt ved ligninger og uligheder. Som problemdomæne benyttes elektriske kredsløb, idet variablerne bl.a. er strømstyrke, spænding og modstand, og bindingerne er udtryk for elektriske love, f.eks. Ohm´s lov. Projektet går ud på at implementere et tilsvarende sprog, eller dele heraf, i form af en Java-pakke.

[1] G. J. Sussmann and G. L. Steele,
CONSTRANTS - A Language for Expressing Almost Hieararchical Descriptions,
Artificial Intelligence, Vol. 14, No. 1, 1980 (pp. 1-39).

15. En Java-pakke til regning med tynde matricer (modul 1)

Dette projekt går ud på at konstruere et program til regning med såkaldte *tynde* matricer, dvs. matricer, hvor relativt mange af elementerne er nul. Tynde matricer forekommer i mange praktiske beregningsopgaver. Da der ofte er tale om meget store matricer, er der behov for, at de datastrukturer, som anvendes til repræsentation af matricerne, er effektive med hensyn til pladskrav.

I nedenstående reference [1] er beskrevet forskellige datastrukturer, som med fordel kan benyttes til repræsentation af tynde matricer. Der skal konstrueres et generelt program til regning med tynde matricer, som indeholder de almindeligst forekommende matrixoperationer: addition, multiplikation, inversion, samt faciliteter til indlæsning og udskrivning af matricer. Alternativt kunne projektet rette sig mod C^{++} i stedet for Java.

[1] J. K. Reid. Data structures for sparse matrices.,
J. K. Reid (Ed.).
The relationship between numerical computation and programming languages,
North Holland, 1982.

[2] D. J. Rose (Ed.).,
Sparse Matrix Computations.
Academic Press, 1975.

16. En Java-pakke til regning med uendelige potensrækker (modul 1)

Dette projekt hører til det datalogisk/matematiske område, der kaldes *symbolsk formelmanipulation.*

Potensrækker benyttes til at opnå approksimative løsninger til en lang række problemer inden for anvendt matematik. Eksempelvis kan funktionen cos(x) approksimeres ved den *endelige* potensrække $1 - x^2/2 + x^3/3$. Denne potensrække er fremkommet ved af afkorte en *uendelig* potensrækkeudvikling for cos(x) og kan f.eks. benyttes i en datamaskine til numerisk beregning af cos(x) for givne x-værdier.

Regning med endelige potensrækker giver imidlertid ofte numeriske problemer, og det er derfor ønskeligt i stedet for at kunne arbejde med uendelige potensrækker.

Projektet går ud på at konstruere og afprøve et program til regning med uendelige potensrækker. Der kan tages udgangspunkt i den implementation, der er beskrevet i [1].

[1] S. Harrington,
Infinite Power Series,
Software - Practice and Experience, Vol. 10, 1980 (pp. 835-848).

17. En Java-pakke til regning med polynomier (modul 1)

Projektet går ud på at stille faciliteter til rådighed, i form af en Java-pakke, som gør det muligt at foretage symbolske beregninger på polynomier (addition, multiplikation, differentiation osv.). I reference [1] er angivet en mulig datastruktur til repræsentation af polynomier i flere ubekendte.

[1] D. E. Knuth,
The Art of Computer Programming, Vol. 1, Fundamental Algorithms,
Addison Wesley, 1973 (2. edition) (pp. 355-359).

18. Et krydsreferenceprogram til Java (modul 1)

Et krydsreferenceprogram er et meget nyttigt værktøj ved programmering og programvedligeholdelse, idet det kan give et overblik over anvendelsen af de brugerdefinerede navne i et program. Et krydsreferenceprogram kan f.eks. producere en liste over samtlige variabler i et program med henvisning til de programlinjer, hvori de bliver refereret.

[1] R. S. Scowen,
Some Aids for Program Documentation,
Software - Practice and Experience, Vol. 7, No. 6, 1977 (pp. 779-792).

19. Indføring af et mængdebegreb i Java (modul 1)

Projektet går ud på at konstruere en Java-pakke til "regning" med mængder. Klassen skal bl.a. muliggøre repræsentation af mængder og indeholde de mest almindelige mængdeoperationer, såsom bestemmelse af fællesmængde og foreningsmængde. Desuden skal der gives eksempler på praktisk anvendelse af den implementerede klasse.

[1] W. A. Wulf, M. Shaw, P. N. Hilfinger and L. Flon,
Fundamental Structures of Computer Science,
Addison Wesley, 1981 (pp. 285-288, 483-528).

[2] K. Kohel and U. Masctera,
KOMA - A Conceptual Set Model - Implemented with SIMULA as a Semantic Net,
Proceedings of the 14. Simula Users´ Conference, Stockholm, August 1986.

20. Et sprog til håndtering af grafstrukturer (modul 1 og 2)

Inspirationen til dette projekt stammer fra artiklen [1], hvori beskrives et sprog, GRAPHIX, til håndtering af grafstrukturer. Projektet går ud på at programmere dette sprogs faciliteter samt foretage en vurdering heraf på baggrund af en række anvendelseseksempler.

[1] G. Sutcliffe,
GRAPHIX - a Graph Theory Sub-Language,
Int. J. Computer Math., Vol. 17, 1985 (pp. 275-296).

21. Skiplister (modul 1)

En skipliste er en datastruktur, som kan benyttes som et alternativ til balancerede binære træer. Skiplisters effektivitet modsvarer AVL-træers. Det gennemsnitlige tidsforbrug er i begge tilfælde O(log n) for indsættelse, søgning og sletning. Empiriske undersøgelser har vist, at implementeringer, som benytter skiplister er lige så effektive, og i nogle tilfælde mere effektive, end implementeringer, der benytter AVL-træer (selv ikke-rekursive implementeringer af AVL-træer). Hertil kommer, at skiplister synes langt simplere at implementere end AVL-træer.

I dette projekt implementeres og afprøves en Java-pakke, der realiserer en abstrakt datatype ved hjælp af skiplister. Datastrukturen er udviklet af W. Pugh, som har beskrevet både dens implementering og effektivitet grundigt i [1].

[1] W. Pugh,
Skip lists: A probabilistic alternative to balanced trees.
Communications of the ACM, 33(6), June 1990.

22. Løsning af ligningssystemer (modul 1)

Løsning af ligningssystemer er et hyppigt forekommende problem i forbindelse med naturvidenskabelige problemstillinger.

Ofte er der tale om *lineære* ligningssystemer (systemer på formen Ax = b, hvor A er en kvadratisk matrix, og x og b er vektorer). I sådanne tilfælde kan en løsning bestemmes ved brug af et tilgængeligt numerisk programbibliotek. Det kræver sædvanligvis en smule programmeringskendskab, men er forholdsvist simpelt.

Hvis ligningssystemerne derimod *ikke* er lineære (hvis der f.eks. indgår produkter af de ubekendte), kan det være vanskeligt at finde programmel, som kan benyttes til løsning. I sådanne tilfælde kan det derfor blive nødvendigt selv at udvikle en algoritme til formålet.

I artiklen [1] er beskrevet en simpel algoritme til løsning af såvel lineære ligningssystemer som visse typer af ikke-lineære ligningssystemer. Dens styrke i forhold til andre tilsvarende algoritmer hævdes i artiklen at være dens simpelhed og hastighed.

Målet med projektet er at implementere og vurdere den angivne algoritme. Er de i artiklen fremførte påstande korrekte? Kan algoritmen forbedres?

[1] E. Derman and C. J. Van Wyk,
A Simple Equation Solver and its Application to Financial Modelling,
Software - Practice and Experience, Vol. 14, No. 12, 1984 (pp. 1169-1184).

23. **Kognitiv modellering** (modul 1)

En *kognitiv simulering* er en simulering på en datamat af mentale og kognitive processer. Modellen konstrueres normalt af en psykolog med det formål at opnå en bedre forståelse af menneskelig adfærd, med vægt på forståelsen af de mentale processer, der ligger bagved adfærden. Sigtet med kognitiv modellering er at udtrykke og afprøve teoridannelser om mentale processer.

Dette projekt har til formål at afprøve en klassisk kognitiv model, nemlig den såkaldte EPAM-model, som blev udviklet af Feigenbaum i 1963 [1]. EPAM er en model af menneskelig indlæring af ord uden mening (nonsensord). EPAM præsenteres for en række nonsensord og "husker" ordene ved hjælp af et diskriminationsnetværk. EPAM udviser nogle af de samme træk, som kan observeres hos mennesker, der lærer, nemlig glemsel og forstærkning.

For at kunne afprøve EPAM, skal modellen realiseres i et passende programmeringssprog. Idet der er tale om en relativ gammel model, kunne det være interessant at sammenholde den med eventuelle nyere modeller.

[1] E. A. Feigenbaum,
The Simulation of Verbal Learning Behaviour,
E. A. Feigenbaum and J. Feldman (eds.),
Computers and Thought. McGraw-Hill, 1963 (pp. 297-309).

[2] H. A. Simon and E. A. Feigenbaum,
An information-processing theory of some effects of simmilarity, familiarization, and meaningfullness in verbal learning,
J. Verb. Learn. Behav., Vol. 3, 1964 (pp. 385-396).

[3] D. L. Hintzman.,
Explorations with discrimination net model for paired-associate learning,
J. Math. Psychol., Vol. 5, 1968 (pp. 123-162).

[4] E. A. Feigenbaum and H. A. Simon,
EPAM-like models of recognition and learning,
Cogn. Sci., Vol. 8 (4), 1984 (pp. 305-336).

24. Et værktøj til udvikling af ekspertsystemer (modul 1)

Et ekspertsystem er et edb-baseret system af viden inden for et afgrænset fagområde. Systemet realiseres sædvanligvis ved at udfritte en eller flere sagkyndige på området (eksperterne) om deres viden. Denne viden lagres i en datamat på en sådan form, at den kan bruges til at besvare spørgsmål i relation til det faglige område. For eksempel kan lægers viden om symptomer på sygdomme lægges ind i et ekspertsystem, som derefter vil kunne bruges til at stille patientdiagnoser. Lægers ekspertise på et specifikt område er således gjort tilgængelig for andre i form af et program. Programmet kan bruges af ikke-eksperter som hjælp ved diagnosticering. På denne måde kan en eventuel mangel på eksperter afbødes. Faktisk kan systemet i nogle tilfælde vise sig mere kompetent end en enkelt ekspert, idet dets viden kan være indhentet fra mange uafhængige eksperter.

Hvorvidt man ønsker helt at erstatte eksperten med et ekspertsystem må vurderes i de enkelte tilfælde. Manglen på læger i Afrika har for eksempel medført, at ekspertsystemer benyttes til at stille diagnose i forbindelse med en del specielle sygdomme, blandt andet malaria.

Ekspertsystemer anvendes i dag med succes inden for en lang række af fagområder, spændende lige fra medicin, kemi og geologi til jura, politik og økonomi.

Et ekspertsystem består af følgende tre komponenter:

(1) Vidensbasen, hvori der ligger fakta og regler. Eksempel på et faktum: “Det er regnvejr”. Eksempel på en regel: “Hvis svalerne flyver lavt, så bliver det regnvejr”. Basen kan desuden indeholde information om sikkerheden af de lagrede fakta og regler.

(2) Inferensmaskinen, som er i stand til at drage slutninger ud fra vidensbasen indhold, f.eks. besvare spørgsmål som “Er det regnvejr?” og “Hvilke årsager kan der være til, at det er regnvejr?”.

(3) Brugergrænsefladen, som benyttes til at udveksle information med systemets brugere, bl.a. at indlæse spørgsmål, at udskrive svar, at udskrive begrundelser for svar og at modtage anvisninger om at revidere vidensbasens indhold.

I dette projekt implementeres og afprøves en Java-pakke til udvikling af ekspertsystemer (en såkaldt “ekspertsystemskal”). Faktisk finder der allerede et sådan program, men det er skrevet i sproget Modula-2 (en videreudvikling af Pascal). Dette program (ca. 500 programlinjer) kan passende benyttes som udgangspunkt, men det vil ved programmeringen i Java være oplagt at anvende objektorienteret programmering.

[1] B. Sawyer and D. Foster,
Programming Expert Systems in Modula-2,
Wiley Press, 1986.

25. Fraktionelle kaskader (modul 1 og 3)

Fraktionelle kaskader en navnet på en datastruktur, der med fordel kan anvendes i forbindelse med databaser, der indeholder geometrisk information (f.eks. geografiske databaser). I projektet undersøges og afprøves denne datastruktur nærmere. Projektarbejdet kan muligvis resultere i udformning af kernen til et databasesystem, der baserer sig på datastrukturen.

[1] B. Chazell and J. L. Guibas,
Fractional Cascading: I. A Data Structuring Technique,
Algorithmica, Vol. 1, No. 2, 1986 (pp. 133-162)

[2] B. Chazell and J. L. Guibas,
Fractional Cascading: II. Applications,
Algorithmica, Vol. 1, No. 2, 1986 (pp. 163-191)

[3] K. Mehlhorn and S. Näher,
Dynamic Fractional Cascading,
Algorithmica, Vol. 5, No. 2, 1990 (pp. 215-241).

26. Problemløsning (modul 1 og 3)

R. E. Korf beskriver i en artikel [1] en metode til bedste-først søgning, som er effektiv med hensyn til pladsforbrug. Projektet går ud på at undersøge algoritmen og indlejre den i et generelt system til problemløsning.

[1] R. E. Korf,
Linear-space best-first search,
Artificial Intelligence, Vol. 62, No. 1, 1993 (pp. 41-78).

27. Et generelt program til fremstilling af eventyrspil (modul 1)

I nedenstående reference [1] er demonstreret, hvorledes man kan udvikle sine egne udgaver af det såkaldte eventyrspil. Programmeringen er imidlertid foretaget i BASIC, hvilket vanskeliggør programoverskueligheden og mindsker programmellets fleksibilitet betydeligt. I dette projekt undersøges Javas egnethed som alternativt implementeringssprog.

[1] N. Søndergaard,
Lav dine egne computereventyr med BASIC.
Borgen, København 1985.

28. Maskinindlæring (modul 1 og 3)

Maskinindlæring omhandler automatiseret indlæring af begreber og regler på baggrund af erfaring. Området har f.eks. stor praktisk betydning ved udvikling af regelbaserede ekspertsystemer, idet etablering af en vidensbase er flaskehalsen i udviklingssprocessen.

I dette projekt implementeres og afprøves en forholdsvis ny algoritme [1], kaldet CN2, der kan bruges til at danne klassifikationsregler ud fra repræsentative eksempler. Som et eksempel på anvendelse kan nævnes skelnen mellem kræftfremkaldende og ikke-kræftfremkaldende stoffer.

Hvis tiden tillader det, foretages en sammenligning med den hidtil mest benyttede algoritme til maskinindlæring, nemlig ID3 [2].

[1] P. Clark and T. Niblett,
The CN2 Induction Algorithm,
Machine Learning, Vol. 3, No. 4, 1989 (pp. 261 - 283].

[2] J. R. Quinlan,
Induction of Decision Trees,
Machine Learning, Vol. 1, No. 1, 1986 (pp. 81-106).

[3] T. G. Dietterich, R. London, K. Clarkson and G. Dromey,
Learning and inductive inference,
P. R. Cohen and E. A. Feigenbaum (Eds),
The Handbook of Artificial Intelligence, Vol. 3, Pitman, 1982 (Ch. XIV).

29. En programmelkerne til datamatstøttet undervisning (modul 1)

Med udgangspunkt i nedenstående referencer, primært [1], konstrueres en programmelkerne til datamatstøttet undervisning.

[1] K. Ahmed, D. Ingram and C.J. Dickinson,
Software for Educational Computing,
MTP Press, England 1983.

[2] I. Boserup and J. Feder,
Bogen om MikroTutor,
Museum Tusculanums Forlag, København 1984.

30. En Javagrænseflade til relationsdatabaser (modul 1 og 2)

Projektet går ud på, med inspiration fra bl.a. nedenstående referencer, at udvide Java med faciliteter til brug af relationsdatabaser.

[1] M. Bever and J. Lockemann,
Database Hosting in Strongly-Typed Languages,
ACM. Trans. Database Systems, Vol. 10, No. 1, 1985 (pp. 107-126).

[2] S. Alagi'c and A. Kulenovic,
Relational Pascal data base interface,
The Computer Journal, Vol. 24, No. 2, 1981 (pp. 112-117).

[3] E. Horowitz and A. Kemper,
AdaREL: a relational extension of Ada,
Computer Science Dept. of Southern California, Los Angeles, November 1983.

[4] S. Alagi'c,
Object-oriented database programming,
Springer Verlag, 1988.

31. Funktionsorienteret programmering (modul 2 og 3)

I nedennævnte bog af Henderson (1) gives en god introduktion til funktionalprogrammering, dvs. programmering ved hjælp af funktioner, og der beskrives et funktionsorienteret sprog, der er en variant af Lisp. Bogen angiver endvidere, hvorledes sproget kan realiseres i et system, Lispkit, bestående af en oversætter og en fortolker. Projektet går ud på af implementere Lispkit ud fra bogens anvisninger.

[1] P. Henderson,
Functional Programming. Application and Implementation,
Prentice-Hall, 1980.

[2] B. Klitgaard og B. F. Mortensen,
FLISP - en Lisp-fortolker i Simula,
RUC-rapport, datalogi, 1985.

32. Grammatikker for formelle sprog (modul 2)

Formålet med dette projekt er at studere de grundlæggende egenskaber ved grammatikker for formelle sprog med henblik på konstruktion af effektive algoritmer til syntanksanalyse. Projektet tænkes at resultere i et system, f.eks. skrevet i Java, til håndtering og egenskabsbestemmelse af grammatikker. Et sådan system vil bl.a. være relevant i forbindelse med konstruktion af et generelt system til syntaksanalyse, en såkaldt "generel parser", eller til parser-generering, dvs. automatisk generering af en parser ud fra en given grammatik.

Nedenfor gives en række eksempler på, hvad det i projektet konstruerede system tænkes at kunne:

- afgøre hvorvidt en grammatik repræsenterer et ikke-tomt sprog
- fjerne utilgængelige og nytteløse symboler
- fjerne tomme produktioner samt enkeltproduktioner
- eliminere venstre-rekursion
- konvertere en grammatik til Chomski- eller Greibach-normalform
- konstruere en parse-tabel for en LL(1)-grammatik.

Som udgangspunkt for projektet kan for eksempel benyttes lærebogen af Aho og Ullman [1], hvori algoritmer til løsning af ovennævnte opgaver er angivet. Endvidere kan anbefales Backhouse´s bog om syntaks for programmeringssprog [2], der er knap så matematisk.

[1] A. V. Aho and J. D. Ullman,
The Theory of Parsing, Translating and Compiling. Volume 1: Parsing,
Prentice-Hall, 1972 (pp. 138-163, 334-361).

[2] R. C. Backhouse,
Syntax of Programming Languages,
Prentice-Hall, 1979.

33. Objektorienteret logikprogrammering (modul 2 og 3)

Objektorienteret programmering og logikprogrammering er to programmeringspararadigmer, der i de senere år er blevet meget populære. Objektorienteret programmering understøttes af sprog som Simula, C++ og Java, mens logikprogrammering understøttes af sprog som Prolog og Gödel.

De to paradigmer har hver deres styrke, og det er derfor relevant at undersøge, om det er muligt at konstruere programmeringssprog, som understøtter begge paradigmer. Et bud på et sådan sprog er Prolog++, som beskrevet i [1]. Prolog++ udvider logikprogrammeringssproget Prolog med et objekt- og klassebegreb. På denne måde tilføres Prolog de fordele, der kendes fra objektorienteret programmering, bl.a. modularitet og kodegenbrug.

Der er flere mulig projekter inden for dette felt. Nedenfor ses nogle forslag.

(1) Implementering i Prolog++ af et værktøj til udvikling af vinduesbaserede grafiske brugergrænseflader.

(2) Evaluering af Prolog++.

(3) Sprog til kombineret objektorienteret og logikbaseret programmering (speciale).

[1] C. Moss,
Prolog++ - The Power of Object-Oriented and Logic Programming,
Addison Wesley, 1994.

34. Maskinindlæring ved begrebsmæssig klyngedannelse (modul 3)

[1] R. S. Michalski and R. E. Stepp,
Learning from Observation: Conceptual Clustering.
R. S. Michalski, J. G. Carbonell and T. M. Mitchell (Eds),
Machine Learning,
Tioga, Palo Alto, 1983 (pp. 331-363).

35. Vidensrepræsentation med semantiske net (modul 3)

[1] S. E. Fahlman,
NETL: A System for Representation and Using Real-World Knowledge,
MIT Press, 1979.

[2] P. R. Cohen and A. E. Feigenbaum,
The Handbook of Artificial Intelligence, Vol. 1,
Pitman, 1982 (Ch. III).

36. Ikke-monoton logik (modul 3)

[1] J. Doyle,
A truth maintenance system,
Artificial Intelligence, Vol. 12, 1979 (pp. 231-272).

[2] J. de Kleer,
An Assumption-based TMS,
Extending the ATMS,
Problem Solving with ATMS,
Artificial Intelligence, Vol. 28, No. 2, 1986 (pp. 127-224).

37. Unifikationsalgoritmen (modul 3)

Dette projekt omhandler en ganske bestemt algoritme, den såkaldte unifikationsalgoritme, som er kernen i ethvert system til automatisk bevisførelse, således også i implementationer af logikprogrammeringssproget Prolog. Algoritmen benyttes, kort fortalt, til hvad der på engelsk betegnes "pattern matching", dvs. sammenparring af mønstre, idet der ved et mønster forstås et symbolsk udtryk. Algoritmen bestemmer for to givne mønstre, hvilke variabelsubstitutioner, der vil kunne gøre mønstrene identiske.

Effektiviteten af unifikationsalgoritmen er meget afgørende for den totale effektivitet af et logiksystem, hvilket har bevirket, at der er blevet gjort store bestræbelser på at finde en så effektiv udgave af algoritmen som overhovedet muligt.

Dette projekt går ud på at foretage empiriske undersøgelser af en særligt lovende udgave af algoritmen, der er beskrevet af Corbin og Bidoit [1], og eventuelt fortage sammenligninger med andre udgaver, f.eks. den af Robinson beskrevne [2].

[1] J. Corbin and M. Bidoit,
A rehabilitation of Robinson´s Unification Algorithm,
Information Processing, 1983 (pp. 909-914).

[2] J. A. Robinson,
A machine-oriented logic based on the resolution principle,
Journal ACM, Vol. 12, 1965 (pp. 23-41).

38. Automatisk bevisførelse (modul 3)

[1] J. K. Siekmann and G. Wrightson (Eds),
Automation of reasoning, Vol. 1 and Vol. 2,
Springer Verlag, 1983.

[2] P. H. Larsen,
En Simula-klasse til automatisk bevisførelse,
RUC-rapport, datalogi, 1977.

[3] K. Helsgaun,
Automatisk bevisførelse,
Specialerapport, Københavns Universitet, 1973.

[4] R. J. Cunningham and S. Zappacost-Amboldi.
Software Tools for First-Order Logic.
Software - Practice and Experience, Vol. 13, 1983 (pp. 1019-1025).

[5] C. Walther,
A Mechanical Solution of Schubert´s Steamroller by Many-Sorted Resolution,
Artificial Intelligence, Vol. 26, 1985 (pp. 217-224).