6/86

Johannes Hansen

SIMULATION AND AUTOMATION OF LEGAL DECISIONS

NORIS (57)

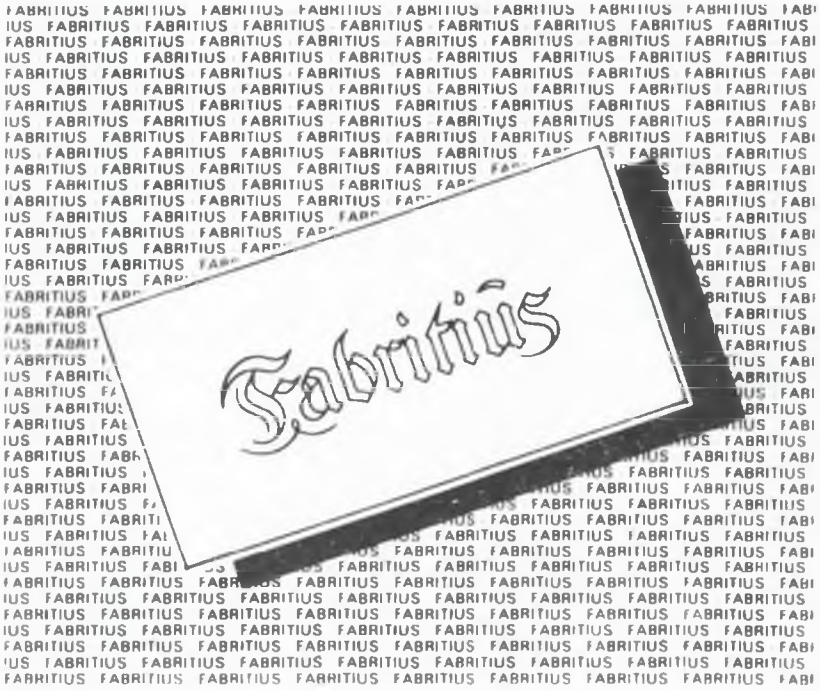
COMPLEX

NORWEGIAN RESEARCH CENTER FOR COMPUTERS AND LAW

NORWEGIAN UNIVERSITY PRESS

IBM Bergen: Dreggsalmenning 10/12, tlf. (05) 31 55 00
IBM Stavanger: Auglendsdalen 81, tlf. (04) 58 85 00
IBM Trondheim: Kongensgt. 60, tlf. (07) 53 06 44
IBM Oslo: Dronning Maudsgt. 10, tlf. (02) 20 54 50

FORLAG
GRAFISK INDUSTRI



DATASATS
ELEKTRONISK TRYKKING
INFORMASJONSDISTRIBUSJON

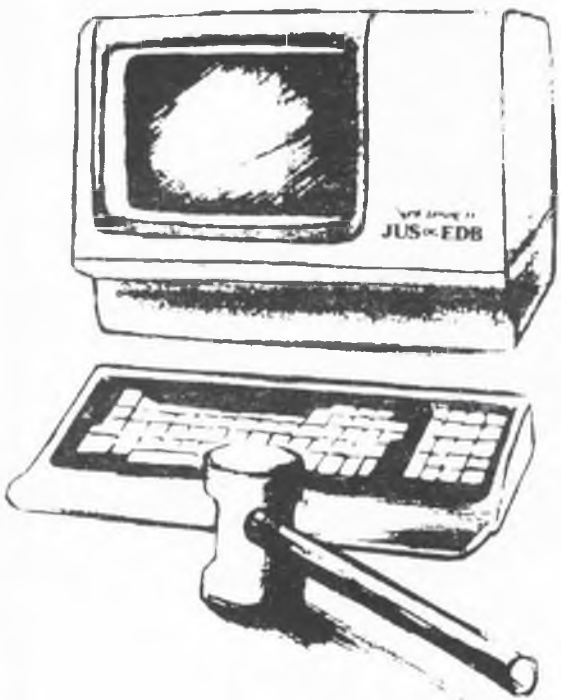
FABRITIUS A S
BROBEKKVN 80
P BOKS 1156 SENTRUM
0107 OSLO 1

TELEFON (02) 64 08 88
TELEX 18 137 fabrin
TELEFAX (02) 64 32 83

Norsk forening for
JUS OG EDB

Postboks 7557, Skillebekk, OSLO 2

Postgiro 5 13 96 54 Bankgiro 8200 42 49727
Postgiro CompLex 2 12 16 63



Foreningen står bl.a. for salget av CompLex-heftene og vedlikeholder abonnementsordningene for serien.

SamvirkendeDataSystemerSamvirkendeDataSystemer
DataSystemerSamvirkendeDataSystemerSamvirkende
SystemerSamvirkendeDataSystemerSamvirkendeData
SamvirkendeDataSystemerSamvirkendeDataSystemer
DataSystemerSamvirkendeDataSystemerSamvirkende
SystemerSamvirkendeDataSystemerSamvirkendeData
SamvirkendeDataSystemerSamvirkendeDataSystemer
DataSystemerSamvirkendeDataSystemerSamvirkende
SystemerSamvirkendeDataSystemerSamvirkendeData
SamvirkendeDataSystemerSamvirkendeDataSystemer
DataSystemerSamvirkendeDataSystemerSamvirkende
SystemerSamvirkendeDataSystemerSamvirkendeData
SamvirkendeDataSystemerSamvirkendeDataSystemer
DataSystemerSamvirkendeDataSystemerSamvirkende
SystemerSamvirkendeDataSystemerSamvirkendeData
SamvirkendeDataSystemerSamvirkendeDataSystemer
DataSystemerSamvirkendeDataSystemerSamvirkende
SystemerSamvirkendeDataSystemerSamvirkendeData

Samvirkende Data Systemer

Statens DataSentral a.s

CompLex nr. 6/86

Norwegian Research Center for Computers and Law
Oslo University
Niels Juels gate 16
0272 OSLO 2

Johannes Hansen

SIMULATION AND AUTOMATION OF LEGAL DECISIONS

NORIS (57)

The project has been supported by the
Norwegian Research Council of Sciences and the Humanities

Norwegian University Press A/S
Oslo

ISBN 82-00-07833-7

Utgivelsene i skriftserien CompLex støttes av:
Den norske Advokatforening
Den norske Bankforening
Digital Equipment Corporation A/S
Ericsson Information Systems A/S
IDA, Integrert Databehandling a.s
Industriforbundets Servicekontor
Norsk senter for informatikk A/S
Norsk Arbeidsgiverforening
Norges Forsikringsforbund
Norsk Hydro
Televerket

Printed in Norway
by GCS A/S, Oslo

PREFACE

In 1981, the Norwegian Research Center for Computers and Law initiated a project to design a relative general model to simulate norms and facts, and to automate some types of legal decisions. The project was named SMARN (a Norwegian acronym for a language for modelling legal norms) and had the project reference NORIS (57) in the NRCCL project classification.

The activity took place based on several years of work at the NRCCL with the problem of modelling discretionary decisions.

In the beginning of the project, the work has as an objective to design a tool for consequential analysis of a set of rules or a set of amendments to rules, leading to the specification of SMARN1 (Chapter 3).

Later, the emphasis was put on automation of decisions. parts of the Inheritance Act was described in SMARN after a major respecification had taking place, resulting in SMARN2 (Chapter 5). This example will be published by Espen Ødegaard in part F of the documentation of the seminar "Modelling Facts, Norms and Concepts", taking place at the NRCCL winter and spring 1984.

This report is part E of the same documentation. The other four contributions have been published in CompLex 8/85.

The discussion of the respecification is given in Chapter 4.

Chapter 1 contains a discussion of Jon Bing's general model of legal problem solving with some proposed amendments.

In Chapter 2 the degree of formalization of model of legal problem solving is increased stepwise. The purpose is to show which restrictions SMARN have, and what its possibilities are, and to stimulate alternative design.

The presented work is colored by the fact that a third level programming language is used to express norms, with the following strong restrictions. The extensions in the SMARN language are modest compared to these restrictions, but contain tools to handle discretionary decisions, sets, a new kind of objects as well as operations on sets and objects. A mechanism to handle schemata is also introduced.

It has been a challenge to direct the SMARN design activity at the NRCCL. The reason probably most important for this challenge, is that the degree of formalization in the theory of norms is relatively low. It should also be admitted that one has been somewhat too eager to implement the formalism rather than discussing their adequacy.

Rather than to implement SMARN2 today, which would take a few man-years, NRCCL should give priority to consider language primitives and structures, as well as logic for actions, actors, time, various norm types etc.

Having had the privilege to work at the NRCCL for 9 years, I here would like to thank the many research assistants and students contributing to the stimulating atmosphere. In special I will give honor to the director, Jon Bing, and the chairman, Knut S. Selmer, for their way of encouraging inexperienced researchers.

I am grateful to Torstein Eckhoff, Sverre Spurkland, Knut S. Selmer, Gert-Fredrik Malt, Jon Bing and Andrew Jones for having read my manuscript and offered comments. I am also grateful to Andrew Jones for having proof-read the English manuscript.

At last I would like to thank the Norwegian Research Council for Sciences and the Humanities for having financed the project through a period of three years.

Oslo, August 21st, 1986

Johannes Hansen

Table of contents

1 SOLVING LEGAL PROBLEMS

4 AMENDMENTS FROM SMARN1 TO SMARN2

1. SOLVING LEGAL PROBLEMS

1.1 Introduction

At The Norwegian Research Center for Computers and Law is running a project with the ambitious goals of modelling facts and norms, enabling a computer to solve some legal problems. This is an experimental effort, rather than an attempt to get rid of the lawyers and judges. The project has been called SMARN, a Norwegian acronym for A Language for Modelling Legal Norms.

Between January 1982 and 1985 the project was financed by the Norwegian Research Council for Social Studies.

Our theoretical concerns lie in the modelling of norms and facts. The practical need is for tools to describe norms, to analyse, predict (simulate) and execute legal decisions by computer.

Our work has been made possible by developments within both the theoretical and practical areas. The latter are characterized by more regulations from public administration, which is a relatively modern phenomenon, with census and tax collection as exceptions. The development of computers was a precondition for parts of the development of mass administration.

The development of public administration and computers has made it possible to automate decisions. Such automatic systems are used to administer the Norwegian Housing Aid Scheme (Rynning:1976). Also taxation and social security calculations carried out automatically may be called automatic legal decision systems. A common characteristic for these is their routine operations.

However, the development has not reached very far with respect to the automation of legal decisions. The reason for this is partly that legal problems are often rather different from

problems that obviously may be solved by computers. And, partly, the reason is that existing attempts to operationalize legal decisions have their shortcomings. And, finally, a degree of resistance and ambivalence may be met among the research and development staff in connection with these kinds of applications.

Planning systems (including simulation and prediction) are used for various purposes. JUSSIM (Belkin:1972) and SIMAG (Dotterwich:1978, Franzen:1983) are examples of resource analysis programs ("bottleneck" analysis in networks) to be used for organisational studies. SISYFOS (Byrgesen:1975) is an example of a model to illustrate the interplay between taxrates and social security payments for various types of situations, to be used for example when preparing amendments to acts, etc.

Experimental, consultative systems have been developed to describe permitted marriages (Dini:1983) and to give references to literature of interest concerning a given labour law case (Neal:1984).

(Svoboda:1985) gives an overview of planning systems. Systems for analysing decisions are rather rare, but have appeared either as a part of a prediction system (Lawlor:1980) or as a part of statistical packages. SARA (Hansen:1983) is rather unusual as a tool dedicated to the analysis of legal decisions, although these must be binary and discretionary.

There are quite a few formalisms used for the automatic treatment of norms.

Generally, the work with propositional logic within the field of computers and law, with consistency problems and contradictions, wellformedness, simplicity, holes,etc., is important, but somewhat peripheral to what we shall define as our goals. These are interesting themes for qualitative studies of the legislation (Allen:1980). Some steps have also been taken towards establishing legal expert systems using propositional logic (Dini:1983).

Some efforts have been made recently with the application of first order logic to norm modelling (Sergot:1982).

A good deal of work has been done in the field called deontic logic, i.e. the logic of normative systems concerning permissions and obligations (Hilpinen:1971). The practical use of this work so far is rather limited. And the treatment of discretion is and will be lacking in these theories.

From the area of legal theory, we have taken advantage of Eckhoff and Sundby's further development of legal norm theory (Sundby:1974 and Eckhoff:1975). This is our most important legal-theoretic pillar. But their description of the legal system from a system-theoretic point of view is also important.

In particular, there are two international projects that have a purpose comparable to SMARN's. The first is L. Thorne McCarty's TAXMAN-project (McCarty:1977, 1979, 1980a-d, 1981, 1982, 1983). This project places itself in the Artificial Intelligence tradition. Facts are here represented in a semantic network, while the law is represented in a semantic description. The first part of the project, TAXMAN I, was a test of the ideas in the area of the taxation of the reorganisation of firms. TAXMAN II has elaborated the representation problems, especially conceptual representation, regarded by McCarty as the most difficult problem to solve in a consulting system. This is very interesting, but somewhat exotic to us. We will use static concepts in contrast to what is done in TAXMAN II.

The other project is LEGOL (Stamper:1980). LEGOL is one of the largest projects in the field of modelling legal norms for application by a computer. The project is led by Ronald Stamper at the London School of Economics and Political Science. However, it will most probably not be followed up in its latest versions.

The project started as a study of formal organisations. The apparatus and functions of such organisations are to a large extent specified by rules. A central standpoint in the project was that information systems for organisations must reflect this set of rules, both in the design of the database

- which is the information bank in the information system, but especially when manipulating data in the database.

The theoretical interest of our work is mainly related to the development of a weighing model, the development of fact and norm models, modelling operative relations between norms, and the development of language constructions for these in SMARN.

1.2 Delimiting the concept of a legal problem

We are not interested only in developing a formalism to describe legal norms and facts. It is also to be used to simulate and automate the solution of legal problems. Therefore, it is necessary to suggest in some detail what one means by a legal (decision) problem. I will start by listing phenomena which the concept should not contain. Legislative technique represents a first point, analyses of problems to be regulated and expressing rules are others. Theoretical discussion and solutions of theoretical problems are large areas to be kept out. Discussions of legal principles are sorted hereto.

What I do intend to catch are problems presented for solution in public administration or before a court, or for a lawyer.

I will give a tentative definition of the concept of a legal problem. A legal (decision) problem is a problem that can be solved by the execution of a legal decision.

By decision, I primarely refer to a choice between alternatives.

The phrase 'can be solved by the execution of a legal decision' contains a requirement that there exist a legal body competent to solve the actual problem.

For something to be a legal decision, we must check the power of the executive, the sources of law and evidence used in the decision as well as the form in which the decision was made.

If a decision-maker is not empowered to make a decision of the actual kind, I would not call the decision legal.

By 'the sources of the decision', I mean to refer to the facts and the legal sources. Legal decisions should not be made on the basis of rumours, for example. But it is difficult to establish a sharp criterion about proofs to call something a legal decision. Likewise, valid and relevant legal sources should be the sources of the decision.

With respect to the form of the decision, one condition must be that the conclusion is reached, or grounded, according to a legal argumentation standard. Conditions on proof, interpretation, harmonization, subsumption and application of principles of legal sources should be established. These conditions cannot be stated sharply either. Another condition is that the decison is made in a valid form. For example, the jury must be correctly established, a judge must be impartial, certain essential utterances must take place, etc.

Before a problem 'can be solved' it must be raised as a problem and it must be raised to the appropriate legal organ. Before it will be solved, there must in addition exist a duty, or a permission, and a will in the organ to solve the problem.

I shall also use the concept of a legal problem in regard to problems where a solution is not requested. I shall not reserve the concept to a special phase, e.g a phase where the problem is brought to the knowledge of a person with legal power. That is: we have a legal problem where a situation occurs and legal rules may be applied to the situation, given that a solution had been requested.

One should not be distracted by the fact that legal decisions may take different forms. They may take the form of advice (for example from a solicitor), a decision in public administration, by judges etc.

I want to emphasize that the suggested definition says 'can be solved'. Obviously, there exist legal problems where no solution is requested. But if it is, then a legal solution can solve it.

In addition to there existing legal problems that will not

be solved because no solution is requested, I assume that it has happened that non-legal problems have been solved in court.

The tentative definition does not explicitly rule out wrong legal decisions. It only points out the certainty that a solution exists for certain problems. But it may be wrong, legally. Another aspect is that it may then often be appealed.

In the tentative definition, it should also be stressed that the solution may be feasible. Only after the solution is carried out, will the legal problem be solved.

Jon Bing has given a definition of what I regard a similar concept of legal decision problem to the one I am looking for (Bing:1977, p.18). But his definition is rather weak:

> "a legal problem is a problem to whose solution legal argumentation may contribute"

According to this definition, it is a necessary and sufficient condition for a problem to be legal that legal arguments may contribute to its solution. But from this it follows that it is neither necessary nor sufficient that legal argumentation actually contributes.

My suggestion is stronger, since it gives a single necessary and sufficient condition, namely that the problem can be solved by a legal decision.

Must legal arguments always contribute to the solution of legal problems? That is: Is it a necessary condition for something to be a legal problem that legal arguments contribute to its solution? Bing does not raise this question. It is correct to answer yes, given my tentative definition of legal problem. Legal argumentation always contributes when a solution through legal procedure is requested. Accordingly, legal arguments do not contribute when a request is lacking.

A weakness in Bing's definition is that what he calls the total problem, of which only a part perhaps is a legal problem, must also be called legal according to the definition. Since legal argumentation may contribute to solving the partial problem (the legal), it contributes to

solving the total problem.

It is only in one case that legal argumentation is sufficient to solve the problem. That is in a dispute where the parties to a hypothetical case accept the argumentation. Here, it is the parties first and foremost that make the decision that solves the problem, after advice from legal professionals.

Legal problems may be solved in many other different ways than by legal decisions, through the death of a person, by agreement between the parties, etc.

An example of this kind may be that A has offended B in an illegal way. A and B then solve the problem in private, and the problem may cease to exist. But before the solution, a situation existed that might have been given a solution by legal procedure.

The tentative definition of a problem may now be rephrased: a legal problem exists only if a set of legal norms may be applied appropriately by a competent legal organ to a factual situation that has occurred, or as a factual situation occurs.

To characterize the problem, it is not sufficient to describe the facts. For under different legal source situations, or jurisdictions, the problem may be different, although the facts are identical. The same may be said about different legal source principles (principles of interpretation, harmonization, delimiting of sources, etc.). But I shall choose to regard these as constant in the model I develop. In other words, we do operate within the same jurisdicition. This assumption represents another problem, which I shall leave untreated.

A legal problem of type x may be described by a triple (x,Fx,Rx), where x is the general name of the problem, i.e. the characteristic of this type of problem. Fx refers to facts for the problem and Rx refers to the relevant legal sources.

Fx = {f1, f2, ... , fn} , fi refers to fact i.

Rx = {r1, r2, ... , rm} , rj refers to source j.

1.3 Paradigm for the general model of legal problem solving

We shall operate with a model of the real world having three levels:

- a world of things and events,
- expressions about the world (descriptive and normative) in a language form,
- understanding of the world (meanings about ...)

To make models of concepts, facts and norms, one needs to describe in a language some part of nature and society to which one wishes to refer. Let us call such models world models.

Physical reality forms the lower level in every realistic world model.

Decisions and reasons, descriptions of facts, communications between people all take place in some language or other. It is, then, hardly unnatural to separate out oral and written communication as belonging to a particular level in the model: a level for language expressions about the world. The contents of these expressions often exist independently of the source. And the opinion of the source may often be somewhat different from a natural interpretation.

Legal decisions are based on knowledge about the facts of the cases and valid law. Opinions about facts and the law, among those who make the decisions, are quite central for the result of the case. Understanding of the world will therefore be included as the third level in our world model.

Some central processes to be described in this model are: observation, interpretation and expressing.

We may describe these processes as relations between the actor and the world, expressions about the world and understanding of the world. Observation may, with the goodwill of the reader, be applied to all the three levels, interpretation and expressing only to the level of understanding.

Let w0 be a part of the world. Let o,e and i be the processes observe,express and interpret. Then a´s process of

observation, expression and interpretation of w0, expressions of w0 and understanding of w0 may be presented as in figure 1-1.

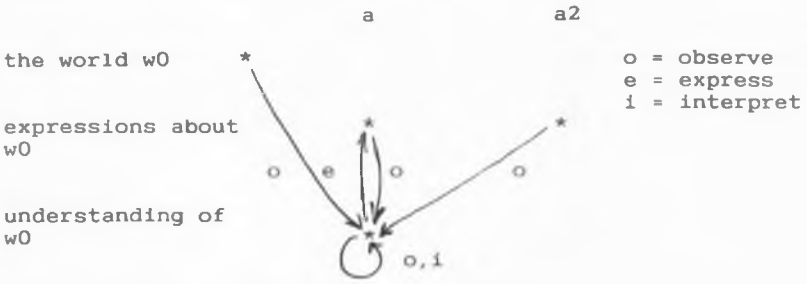


Figure 1-1: A world model with processes seen from a.

Language shall here be understood in a wider sense as a means of communicating meanings (understandings) about the world. It may thus be natural -, picture -, body -, finger-language and many others.

There should exist fixed conventions to interpret the signs in the language.

1.3.1 The problem solving situation

I will characterize the problem solving situation in terms of five factors:

- access to the facts of the case,
- access to the legal sources,
- the problem solver(s),
- available resources,
- other factors.

(1) Access to the facts of the case

Access depends on the mode of search employed. Main types are written and verbal sources, as well as one´s own observations. The availability and reliability of the sources also play a central role.

(2) Availability of the legal sources

Availability depends on where the sources are stored (own mind, library, at colleagues' or in public offices, information systems with and without own access, other libraries etc.). Partly it depends on the structure of the sources as well as on search system. Availability may also be dependent on the volume of the sources.

(3) The problem solvers

I will regard problem solving partly as application of knowledge, partly as learning and partly as making value preferences. Background knowledge may be classified as knowledge about concepts, language, legal sources (including structure), legal method, the case area, etc.

The learning process consists of learning the special case as well as updating knowledge of the other areas mentioned above. Particularly important is the learning of the relevant legal sources. Methods may be own searching, asking colleagues, using text retrieval systems, etc.

The set of moral and other values of the problem solvers may change as a result of the work with the case.

(4) Available resources and other factors

The problem solving situation may be influenced by time pressure, heavy working load and general conditions for the parties involved in the solution of a legal problem. Of particular importance is the situation of the persons who are to solve the problem.

1.3.2 Solving problems

We shall initially describe problem-solving as initiating problem solving, identifying problems and solving identified problems. The initiation process will be described in 1.3.3 while the identification process is described in 1.4.

A very abstract flowchart for the process is given below:

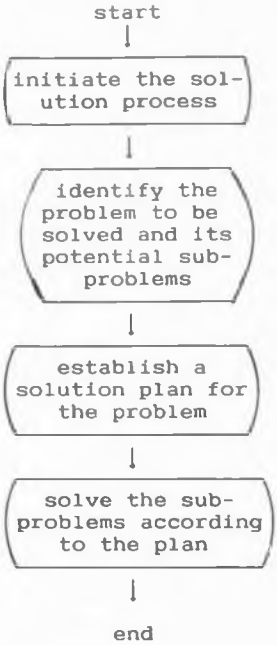


Figure 1-2: Solving legal problems

1.3.3 Initiating the solution process.

This principally consists in a person putting forward a request or an order that some problem solvers should solve a problem, which might be composite.

The problem may be rejected for several reasons, e.g. a lawyer's work pressure. Other forms of problem solving (as in the court) cannot be rejected, only delayed, for such reasons. Generally, we may speak about a preliminary accept to solve a problem. Then it is agreed to initiate the case that ought, can or must, be solved. The purpose of this identification may be to check if some legal problem exists, or to refer the client to a colleague. With a sufficiently exact identification of the problems, a final acceptance or refusal may be made.

We can now say that the problem solution process is initiated. Let us say that a result of the initiation is a

closer definition of which problems are to be solved.

1.4 Identifying legal problems

To identify a legal problem is to answer the question which problem occurs? For a specific problem, x, to occur, certain facts and legal sources must be relevant. The question has therefore two natural subquestions, namely: which facts, Fx , occur and which legal sources, Rx , are relevant for these facts? Specifying such a triple

(x,Fx,Rx)

for a given problem, will be referred to as identifying the legal problem.

It is obviously impossible to identify anything at all in an adequate manner without criteria for the identification. These give conditions for the occurence of a problem.

If a problem P occurs when conditions B are satisfied, this may be expressed:

if B then P.

This is a rule to establish the existence of problems of kind P. Then one could think of rules to express how a given problem should be solved:

if P then L,

where L is the solution of the problem P. But instead of such a detour, rules are commonly expressed on the form

if B then L.

So, when one identifies legal problems, this is done relative to the conditions in the rules. If a condition is satisfied, the problem the norm is introduced to solve occurs. Otherwise not.

To simplify the identification of problems, by simplifying the search for norms, the norms are systematized. The statute rules are systematized in a hierarchic way by laws, chapters, sections, etc. But other systematics would also be possible.

An example is a chronological ordering of enforced sections, without names of laws, chapters, etc. Drastically less efficient, but possible. The point here is only that the norm systematics is just a tool when identifying problems and their corresponding solutions. (The same holds true for the systematics for regulations, directives, instructions, cases and other legal sources). Such a systematics actually contributes to the meaning of the norms.

Since several norms often contribute to the solution of the same problem, one could theoretically collect all norms contributing to the solution of this problem, and do so for all problems, whether the rules are lawrules, from regulations, instructions, cases or other sources. That amounts to saying that one could introduce one systematics for legal problems. It would surely not be especially efficient, since the relative importance of the sources are so different and since the same norm may be relevant to many problems. But nevertheless, it is theoretically possible to establish such a problem systematics.

As I understand it, the identification of legal problems is quite central to the solution of these problems. This process takes place continuously, from the first time the client comes to the lawyer, until the judges have made their decisions, for example. During this complex process, the insight into the problem has probably increased dramatically. The identification is developed into larger detail (not necessarily more precision or correctness) for the lawyer, for example, after his work with the case. The same holds for the judges.

It is, of course, possible that several legal problems may occur within the same total problem. It should lead to the identification of

(y,Fy,Ry) ,etc.

The identification process mainly concerns the investigation of facts and legal sources. My point is that these processes

must be described as strongly dependent on each other. Further, as will be demonstrated later, it must be possible to return to whatever point is needed, at any time until the problem is solved.

If several subproblems occur in the same total problem, one must identify the separate parts. When one is to solve the total problem, the subproblems must be ordered in a proper way. That is, a solution plan should be made. Such a plan is expressed by means of the subproblems and structure primitives for sequence, alternatives, etc. How this is done depends on how rich the solution system is in structure elements.

Likewise, knowledge about the solution system must be reflected in the solution plan, concerning, e.g., deadlines, queues, etc.

It may be worth saying something about the understanding of legal problems. A citizen must have an idea that he might have a legal problem before he seeks assistance by a lawyer. This might be a strong idea. He might be rather sure that the problem is legal. Or he might be a little uncertain: Might there perhaps be some legal problem hidden here? How he has reached his understanding plays a minor role; what is essential is that he has an idea that a legal problem might be present.

Depending on the nature of the problem, various actions may be relevant. Perhaps the step will be to apply for a social service the person is entitled to, to appeal a procedure, to seek a lawyer's advice, etc.

Since the citizen himself may not make any legal decisions, he must communicate his own understanding of the problem to others, let us say the lawyer he unfortunately seeks advice from. The most important thing will be to communicate the factual circumstances. It is then left to the listener, e.g. the lawyer, to clarify his own understanding of the problems, especially to qualify the relevant legal sources. The understanding must then be supported by strong evidence and a good interpretation of sources as well as their relevance.

If the case is being brought to court, evidence must be given for the facts and arguments for the legal sources' contents and relevance. A lawyer will try to establish a norm which gives as good a solution to the problem as possible for his client.

The court's final decision will be authoritative. A decision will be final if it may not be appealed.

An example: Let us say a farmer, Mr. Smith, has a problem. It consists of the fact that one of his neighbours, Mrs. Hill, uses a forest road Mr. Smith has built to Quarrel. Mrs. Hill does this whenever she wants to go fishing in one of the lakes in her own outlying field. Smith has forbidden her to drive on the road, while Hill means that Smith is not in the position to do so. A legal problem, P0, consists in deciding whether Smith may forbid Hill to use the road to Quarrel. This is a problem for Smith. He discusses several alternative solutions, to raise a barrier, to threaten Mrs. Hill and to seek legal advice to have his legal position in the case stated. He is obviously not himself in a position to make a legal decision that solves the problem.

If Smith goes to a lawyer, it is because he is in doubt whether, or quite sure that, the law is on his side. He would not go to a lawyer if he was sure that the law was on Hill's side. In that case, he would probably give up his claim, or try to solve the issue by non-legal means.

Smith's problem, P1, consists in establishing various action alternatives, and in choosing between these. P1 is not a legal problem. Assume that Smith seeks advice from a lawyer. The problem, P2, for the lawyer is to give Smith advice about preferred ways of proceeding. Here there are several possibilities. He may suggest settling things amicably, or that Smith brings the case to court or that he gives up his position and accepts the traffic. A legal problem in P2 is to decide if P0 is a legal problem. If it is not, it obviously cannot be solved by law. If the problem is legal, the lawyer will reason about Smith's legal protection against Hill's traffic. He has to reach a conclusion on this legal question. Although the answer may be positive for Smith, the lawyer does

not have to advise bringing the case to court. The lawyer´s advise may, however, solve the problem if it makes Hill accept her position.

Assume that Hill continues to drive to the lake and that Smith is advised to sue Hill. Before the case is brought to court, certain conditions must be met. This is a separate evaluation to be made by the court administration. Further, administrative tasks by the court have to be carried out before the case can be brought to the court. In the court questions about evidence, interpretations of norms, procedures etc. will be raised, before the case can be solved.

Further, there exists an appeal-right, involving the case being brought to the Supreme Court for a final decision. But even then, under certain circumstances, it might be brought back to a lower court.

We can think of composite legal problems as split into problems that might be solved separately or in a time sequence. An ordering of problems into sequences may be regarded as a legal problem (procedural).

1.5 Bing´s model of problem solving

It is desirable to demonstrate which restrictions apply to SMARN and what assertions it is based on. Demonstrating limitations will hopefully have a valuable pedagogical effect. By describing the transformation from general methods for legal problem solving to SMARN, one may hope for a partial clarification of these limitations. But obviously one also wants to show what possibilities an implemented version of SMARN might represent.

The choice of a general model was rather easy, as, in 1975, the administrative head of the Norwegian Research Center for Computers and Law, Jon Bing, had developed such a model. It was not only developed with the intention of describing the legal decision process, but also to promote the use of legal information retrieval systems, from which some of its weaknesses stem. But it is a general model, and the obvious

choice for a computer scientist at the NRCCL in need of such a model.

1.5.1 Towards the proven facts of a case

The description Bing (Bing:1975 and 1977) gives of problem solving is a kind of process description. The process starts when a client applies to a lawyer for help. He presents a problem, named the total problem. By means of principles of delimitation, the lawyer establishes his own understanding of the legal problem as a part of the whole problem. By means of principles for searching the facts of the case and for proof and evaluation, the lawyer arrives at the probable facts of the case. By assistance of principles for reasonable doubt, the lawyer arrives at the proven facts.

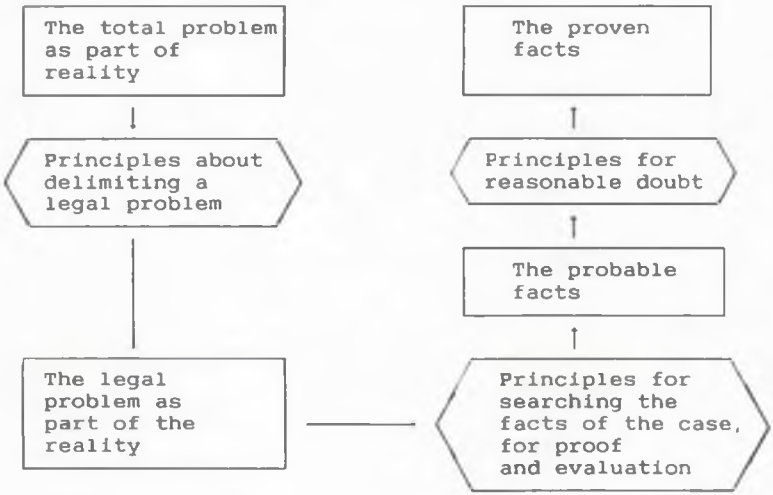


Figure 1-3: From problem to proven facts.

In the figure, rectangles are input and output to processes controlled by the principles represented in the hexagons.

Bing (Bing:1977, p. 20) says:"But generally speaking we may say that up to this point substantive law has not entered into the process".

This is a somewhat unprecise description of the actual

situation. For the proven facts of the case are flavoured by the relevant facts, that is, those facts relevant norms speak about, which indicate, in turn, which legal sources are relevant. Obviously, it would be impractical to check in detail a lot of facts which later (after the selection of sources) prove to be irrelevant. (This should be modified by mentioning the fact that Bing refers to the lawyers background knowledge as the facit for the search of sources, and also that feedback information from later steps might be used).

1.5.2 From proven facts to subsumption

Bing first describes how the total set of relevant legal sources is defined by delimitation principles. Then he describes the availability factors defining the searchable sets of legal sources. (Actually, 'searchworthy' is a better word than 'searchable', since it underlines the cost/benefit analogy).

The search argument is established by the proven facts. When searching the base of legal sources, relevant legal sources are found. After an interpretation process, the relevant norms occur. Comparing proven facts and relevant norms produces legal relevant facts, see figure 2 (Bing:1975, p. 15). This is very unclear from the figure. According to my view, it would be more appropriate to remove the arch between the relevant norms and the proven facts, and let it rather go to legal relevant facts. But this is, as well, incomplete. This imprecision stems from several causes.

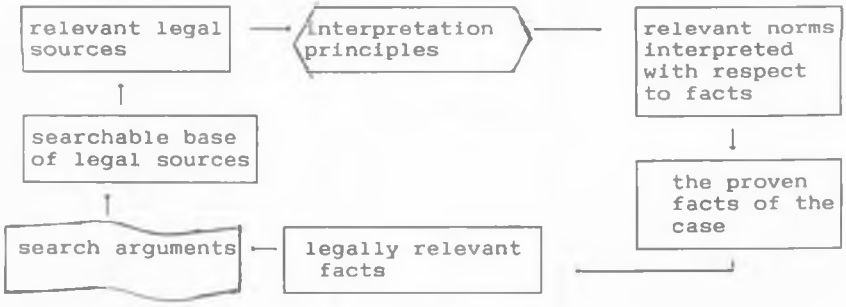


Figure 1-4: The search process.

The most important point is probably that Bing uses only objects in the figures of the type "result of process", but does not represent the processes themselves. The actual example could then have been described as a process called fact-relevance evaluation with relevant norms and the proven facts as input and legally relevant facts as output.

Thereafter, it seems from figures 1 and 7 (Bing:1975) that the proven facts are established once and for all. The figures are imprecise, since on page 17 Bing describes what happens after the relevant norms have been found: "He then goes back to the proven facts of the case (...) to bring forward new, relevant facts, formulates assisting search arguments, etc". Taken literally, figure 7 describes the proven facts of the case fixed once and for all, while the legally relevant facts are just a subset of this set. Other facts cannot then occur as a result of the search process.

The final legally relevant facts and relevant norms occur as a result of an iterative process. "At this stage in the decision process we may say that the lawyer has made a subsumption."

1.5.3 From subsumption to law in force

By interpreting and harmonizing relevant legal sources the normative interval "law in force" is established (Bing:1975, p.18-23).

1.5.4 From law in force to result

The value premises are applied on the normative interval "law in force" and value selected norms occur. Conditions on objectivity are likewise applied on facts, in such a way that legally and objectively relevant facts occur. By application of value selected norms on legally and objectively relevant facts, the result may then be established.

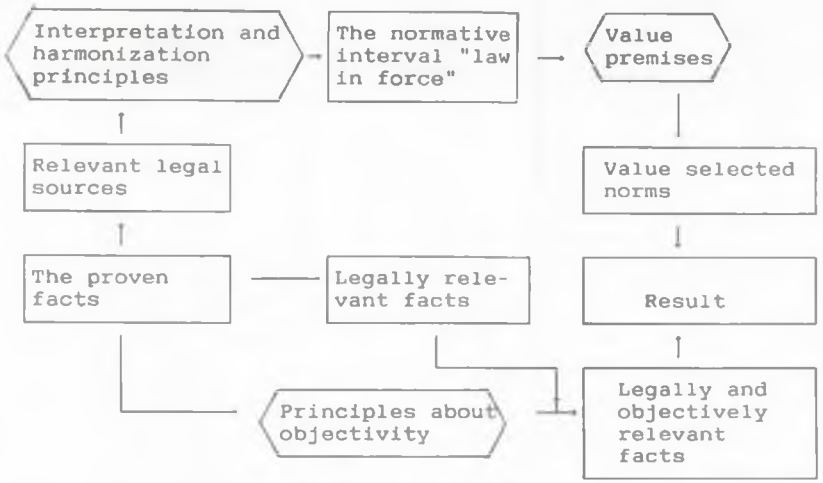


Figure 1-5: The value premises' role and influence on the result.

1.5.5 Evaluating the result - feedback

The value premises may be applied to the result to evaluate it. The premises may influence the judgement concerning relevance of the legal sources.

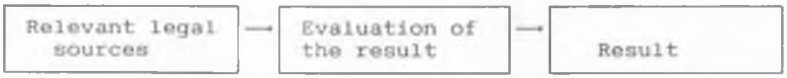


Figure 1-6: Considering a satisfying result

"The consideration of a satisfying result is introduced into the process as a relevant legal source" (Bing:1975,p.28). We observe that through the "consideration of a satisfying result the legal decision model approaches a consequent oriented decision" (p.28).

The process is not ended before the lawyer is "satisfied with the result".

Finally, another important point may be added. The recursive nature of the decision process is not reflected in the model. By that I mean that a process, e.g. a decision, may be

activated as part of a superior decision process. A decision may be activated as a subprocess of an interpretation, which itself is part of a superior decision problem.

Contrary to this, Bing describes the decision process essentially by use of repetitions. This effectively prevents the modelling of very deep decisions, i.e. decisions with several levels of recursion, e.g. several levels of interpretations within interpretations.

A summary of the criticism of Bing's model of the legal decision process:

The development to the point of subsumption is an imprecise description of real processes. It is more intervowen than figure 1 shows (Bing:1975).

The model fails to make explicit an aspect I believe is central, namely the identification of the legal problem.

Processes should have been explicitly represented in the figures.

The recursive nature of the decision process is not reflected.

1.6 Sketch of an alternative model

I will outline a sketch of a model emanating from Bing's. It is amended to some degree especially prior to the subsumption process. The model is presented in figures 1-7 through 1-13. An interview phase WFO (Which Facts Occur) will lead to an isolation of the facts that might have legal relevance, Fx. The search for relevant legal sources WLA (Which Legal sources may be Applied?) is to lead to a reference to possibly relevant legal sources of the case. These legal sources require certain facts, Fh, to be proved applicable. The process WFP (Which Facts are required Proved?) leads to these. If Fh is not included in Fx, then one can choose to search for more facts or reduce the number of relevant legal sources. When Fh is included in Fx, one can say the problem is identified.

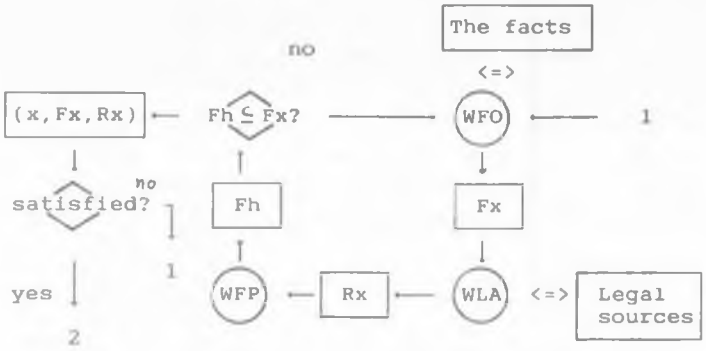


Figure 1-7: The identification process

In figure 1-7 processes are drawn in circles, products (like input and output) in rectangles, while flow of control is represented by the directed, singel-line branches. A product on a control branch is to be regarded as output from the starting-process and input to the ending process of the branch. a ==> b means a operates on b, a <==> b, that both a ==> b and a <== b. Numbers refer to figure 1-13.

1.6.1 On facts

(1) Presenting facts

Without loss of generality we shall name an expressed fact an expression, E, in one or other language, L. It will always be an actor, A, (person) who presents a fact about a situation S1 in a situation S2. He will do this with a certain purpose, P. Presenting facts may then be expressed

A maintains E about S1 in S2 with the purpose P.

(2) Searching facts

Given that a problem, S1, is identified, certain facts describing S1 will be relevant. Searcing facts means trying to

find sources that might produce (put forward) facts. By a source I here mean written, spoken or recorded expressions of facts, the decision-maker himself with his understanding of the facts, or it may be a fact itself. I will illustrate search processes in some more detail.

If one does not have the facts in one's own mind (supposedly then as well), the following sub-processes will be central when searching facts:

Process:	Comment:
1: Define precisely;	What is searched for? (S1)
2: Localize;	Where can I find facts about SI?
3: Transport;	Move oneself or the source to observe S1.
4: Observe;	.. the world, expressions or own understanding.
5: Interpret;	
6: If not satisfied go to 1;	One is presumably satisfied after having found S1 or one gives up.

Figure 1-8: Searching facts

The search takes place on the three levels: in the world itself, in expressions about the world and in understandings of the world, cfr. figure 1-1.

The result of a search process will be regarded as the understanding of facts, that is, interpreted facts (although expressions about facts could also have been regarded as results of a search).

(3) The evaluation of relevance.

Obviously, not all the facts put forward will be relevant. It is up to the relevance evaluation process to determine which of them are.

(4) Evaluating facts

When facts are put forward, one will need to evaluate the various sources' proof-value. One has to judge the content of the expression in the light of how much it pretends to say. There is a considerable difference between mere belief and being quite sure of one's judgement.

One also has to evaluate the reliability of the sources. This includes many factors, from a general evaluation of persons to the distance in time, space, personal relations ... the sources have to what they make statements about.

In the final evaluation one would have to weigh the various proof sources, in order to understand what the facts are, possibly with an additional assessment of the degree of certainty:

A accepts E as a fact about S with certainty C

The evaluative requirements and their form will vary drastically from the solicitor's interview to a proof procedure in court.

(5) Argumentation

Arguing facts will to a large extent concentrate on demonstrating inconsistencies and on finding evidence. The arguing may aim at demonstrating that something held to be a fact is impossible in itself, or that two facts put forward are inconsistent with each other. Or it may aim to convince somebody that a fact is to be accepted as such. Sources of the argumentation are more or less common knowledge (axioms) and rules of inference to be applied to these.

Argumentation influences all the subprocesses.

(6) Monologue and dialogue

When a fact is being searched for in the form of an expression, one must distinguish between two situations. We may call the forms monologue and dialogue. In the monologue,

the actor describes,or argues about, a theme S. While the dialogue may be seen as a question-answering situation including arguing.

We may, somewhat imprecisely, choose to regard the dialogue as a sequence of inter-related monologues.

An important difference is that the dialogue may influence the source, for example, by demonstrating through argumentation a lack of plausibility, an inconsitency etc. Or, more primitively, by focusing on the theme through questioning. The questioning may influence the answerer.

(7) Illustration of the facts of the case

This is a somewhat more detailed illustration of the facts, cf. figure 1-7. We refer to facts as they occur in the world, expressions about facts or understandings of facts. The facts of a case may be on several of these levels.

S1 may be a situation that is central in the case, i.e., still existing, and therefore possible to observe again. Processes from figure 1-8 may be used as a more detailed description of the search for the facts of a case. But how can one find the facts if they are not to be found on the world or expression level? Obviously as understandings, among witnesses, the parties to a case, or others.

Facts in the form of understandings are so important that they deserve special treatment, although it would be possible to overlook this level, since all the understandings that will serve as fact sources except the judges', must be expressed.

The most important reason for a special treatment of understandings is that they may be subject to considerable influence; that is, the facts may be "coloured" when they are expressed.

It is interesting that the facts of the case on the level of understanding take the form of both product and process while, as phenomena in the world and as expressions, they may be regarded as products only. (The facts are described as a product in figure 1-7).

When, for instance, a witness gets a question about a

fact, he will interpret the question, search after facts in his memory, evaluate the relevance of what is found and the value of it. Finally, an answer will be given. This may be specified as below. The states that are established by a process are written within angle-brackets after each process.

Let s be a question about S1 and B the witness´s background knowledge.

Process:	State:
1: a observes s;	<a perceive s>
2: a interprets s;	<a understands s, i.e. u(a,s)>
3: a searches u(a,s) in B;	< a finds r>
4: a evaluates the relevance of r against u(a,s);	<r´, subset of r, is relevant>
5: a evaluates the value of r against u(a,s);	<r",subset of r, is important and valuable enough, etc. for a to be mentioned. B´, Px´>

6: If a is dissatisfied and if it is worth it, a repeats the process from 2,3,4 or 5.

7: a maintains r"

Figure 1-9: Illustration of facts of a case.

Graphically, we may describe the facts of a case on the level of understanding like this:

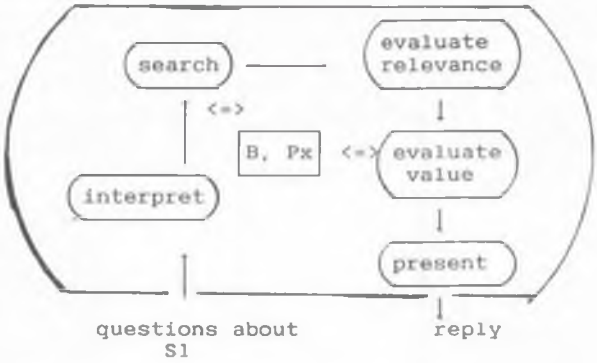


Figure 1-10: Illustration of facts of a case. (Dialogue-situation)

In figure 1-10, B refers to the witness's background knowledge. Px refers to the person's understanding of the problem at hand. ---> represents flow of control and ==> operation on.

(8) Which facts occur?

In the WFO-process (cf, figure 1-7), the actor is the person who investigates the facts. The subprocesses will therefore be relative to this person.

The abstract process WFO is the "driver" in the process consisting of finding relevant facts. It is therefore natural that it differs from the witness in regard to the precision of the identification of the problem (Px) as well as with respect to the actor's role as an interviewer (forming and asking questions). Otherwise, the processes in WFO correspond to the processes described in figure 1-9.

Here is a more detailed illustration of the fact evaluation for the dialogue situation:

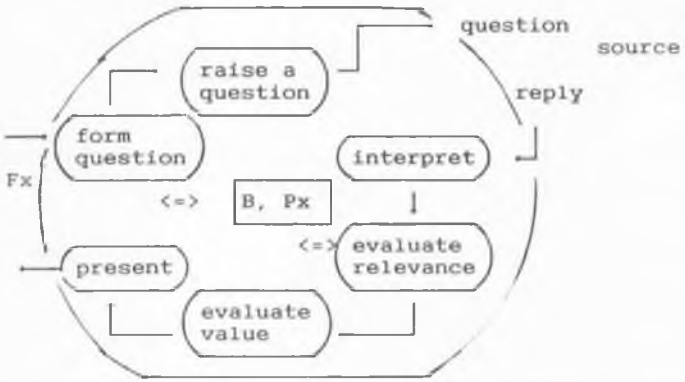


Figure 1-11: Which facts occur? (WFO) (Dialogue situation)

1.6.2 On legal sources

Searching, interpreting, relevance evaluation, harmonizing and evaluating norms, norm expressions or customs are all central

processes when establishing relevant norms in a case.

(1) Searching relevant legal sources and norms.

As soon as relevant facts are identified, various legal sources will be exposed as relevant. The legal sources will have the form

A maintains E about S1 (with purpose P).

The search may take place on all three levels, in regard to customs, expressions (as legal sources) and norms (as understandings).

(2) Interpretation

To establish a norm, norm expressions must be interpreted. This consists of a determination of the concepts occuring in the expression and the meaning of the expression as a whole. The underlying purpose may also be important. The result of the interpretation may be expressed

A has the understanding U of the expression E in situation S (with purpose P)

(3) Evaluation of relevance

When a norm expression is interpreted it must be decided whether it is relevant for the problem at hand.

(4) Harmonization and evaluation

By harmonizing norms is meant a synthesis of norms regulating the solution of the same problem, to yield one norm. The various norms have different sources which will have various weights. In legal method, a formal treatment of differences between laws, regulations, circulars, different levels of court, customs, etc., is established.

The result after harmonization and evaluation will be that

A has the understanding U of sources K1, K2, .. in situation S with purpose P

(5) Argumentation

Here, the argumentation will concern proof of existence, validity and relevance of norms; but weighing in relation to guidelines is also concerned. One may argue that a particular interpretation of a norm expression is correct. The kinds of argumentation are roughly divided into induction (including analogy) and deduction.

(6) Which legal sources may be applied?

A somewhat more detailed illustration of the WLA process is given in figure 1-12. The actor is here the one who is investigating the legal sources.

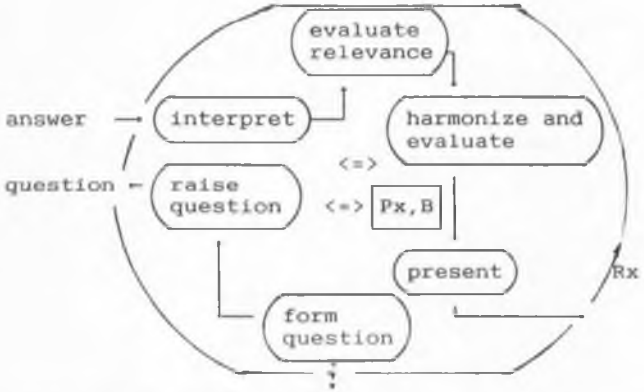


Figure 1-12: Which legal sources may be applied (WLA)?

1.6.3 The model may be applied on different levels of precision.

Problems may be identified on different levels of precision, from a collection of probably relevant facts and legal sources

(e.g. in the form of documents) to the facts and norm-interpretation proved in court. The model is meant to cover all the levels of precision. The variations will best be expressed by different requirements on the processes, according to the background against which the problem identification takes place.

In this model, principles for delimiting legal problems are not reflected, nor principles about searching the facts of the case, proof procedures, evaluation or reasonable doubt.

1.6.4 Solving identified problems

"Solve the problem" refers to the making of the subsumption and the solution of the problem about consequences. That is, it refers to the process of deciding what the consequences are, when the conditions in the antecedent are met. The consequence problem is a problem within the problem; in principle it must itself be identified, etc.

Stated differently, the solution of a problem consists in applying the relevant norm to relevant facts.

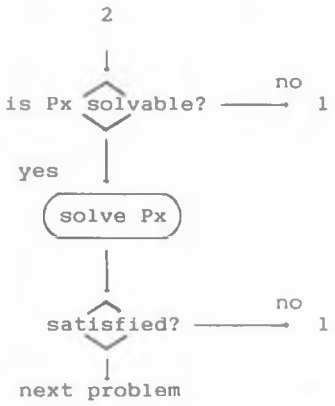


Figure 1-13: Solving the problem (continued from figure 1-7)

The last part of the model follows Bing's closer. I have introduced a question about solvability. This is essentially an investigation of whether or not all the input variables of

the problem are given. If this matter cannot be resolved, one has to return to WFO. In that case the problem may vanish in the light of new insights. Or it may be that other facts or legal sources are found relevant or irrelevant. Consequently, other forms of reidentification may emerge.

1.7 A systematics for problems: Problem hierarchies

What do we really mean when we speak about identification of legal problems? Identified relative to what? When it occurs, a problem is identified by use of criteria defining the existence of legal problems.

Legal classification does not order the problems in a continuum. On the contrary, the problems are classified according to criteria of difference and similarity, and ordered relative to each other. Public law is split into constitutional law, administrative law, penal law and procedural law. Which problems are to be regarded as similar and which as different is a very pragmatic question, strongly dependent on the purpose of the systematics.

This classification and ordering of legal problems might be conceived of as a "problem map", partly mapping problems that have occurred and partly problems that might occur. Since, *a priori*, one is rarely able to predict which problem might occur, a problem map will be dynamic. Problem types arise, develop and disappear.

A reasonable hypothesis is that (legal) problems are best mapped as hierarchies; that is, a class of problems may be split into subclasses, which again may be split. Some problem types may be subclasses of several superior problem types: a problem might be both a tax and a social security problem.

By "hierarchy" I mean a partial ordering of all dependent elements i and j such that either

$i < j$,	(i subordordinate to j)
$i = j$, or	(i and j have equal rank)
$i > j$.	(i superior to j)

If i and j are independent, neither $i < j$, $i = j$ nor $i > j$ is

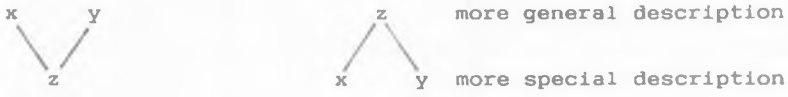
valid.

If i > j and k > j then j is subordinate to both i and k. i > j and k > j is therefore a hierarchy. But if i > j and j > i, then i and j are both subordinate and superior to each other, hence they do not form a hierarchy.

Many writers use the hierarchy synonymously with tree. For trees one has to add one more restriction: if i > j and k > j then either i > k or k > i is satisfied. That is, an element has only one closest superior. Consequently, my hierarchy definition accepts several closest superiors.

Here are the substructure types: a)

b)



(Superior problems are drawn above subordinate).

In the a-case a problem z is both an x and a y problem, while in the b-case both the x and y problem are z problems.

That x is a specific y problem will be denoted

x <u>specific</u> y or (x)y

This is identically stated as

y <u>general</u> x or <y>x.

Let the following abstract systematics be given:

a(b(d),c(e,f(h))),
<d,e>g,
<g,h>i.

(Alternative species are listed with "," between)

The i-problem may be specified on five levels. It is an a-problem, and further both a b and a c-problem. And we can

see that both e and f are special c-problems. For example let a = legal problem, b = private law, c = public law, d = property law, e = penal law, f = administration law, g = compensation law, h = expropriation law and i = problem concerning compensation after expropriation.

The subtypes of an actual problem type should to the largest possible extent cover the whole problem. Thereafter they should have as little as possible in common.

1.8 Characteristics and solvability of legal problems

Problems cannot be solved until they are specified down to a norm level. This means that both facts and legal sources are specified to a norm level. But the problems may be characterized on a higher level than the norm level.

Some words on the relationships between the type and occurence of a problem: in the same jurisdiction a problem-type is determined by the presence of certain types of facts, and the relations between facts. The fact-types act as criteria for the problem-type definition. This, however, is not sufficient if one discusses problem-types from several jurisdictions.

In a given problem the fact-types take specific values, although they might be difficult to decide. Fact-types correspond to problem-types, and values of facts correspond to problem occurrences.

To each type of solvable problem, we may attach specific occurrences of solved problems. Obviously, this may also be done for problem-types that are so general that they cannot be solved, but this is of minor interest.

A solved problem (decision) may become a legal source.

1.9 A solution plan

A lawyer who is to solve a complex legal problem must identify the parts of the problem, that is, define more basic problems. To solve the problem, each part must be solved. And

since problems are often dependent on each other, the order in which they will be solved plays a definite role.

If several lawyers are to be engaged, independent problems may be solved simultaneously.

The solution of a complex problem therefore presupposes a solution plan. Such a plan must contain two types of information, namely an ordering of the problems in a sequence (and in parallel) as well as permitted use of resources for each problem. The types of resources one will describe may be rather comprehensive. Normally, the most important type is staff.

Let us parameterize a solution plan for a problem P like this:

(P,J,t,O).

J is a group of lawyers. t represents time-limitations and O other resources. The O is to act simply as a reminder of important resources other than personnel.

There exist rules about the forming of solution plans. Procedural law contains such rules. The solution plan should obviously satisfy these. Further, a solution of one problem might presuppose others to be solved - and also possibly with specific results. These are often implicit bindings. Sanctions are decided only when the question about guilt is solved positively.

Solution plans will often be fragmental and contain plans only for a few steps ahead. And further progress depends on the results from the problems solved so far.

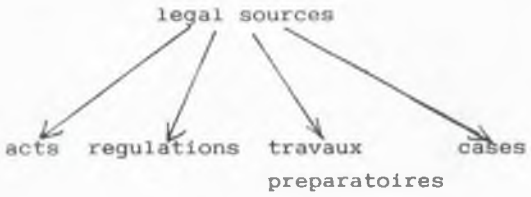
A solution plan may be regarded as an analogy to a very high level computer program.

1.9.1 Composite problems

Problems that often occur in one and the same ordering during the solution of problems may be worth giving a name. It simplifies the references to the complex to use only the name instead of the whole list of ordered partial problems.

1.10 Possible use of the problem hierarchy as a practical categorization

We can think of the problem-map as a map of legal sources sorted on problems,



Directed branches here represent parts of a whole, with the parts specified at the head.

It should be possible to use the problem-map in the law-making phase. That will correspond to a kind of problem sketch. In the beginning of the analysis, before the text is worked out, vague problem characteristics may be the most important characteristics of the problem. As the work proceeds, both the factual conditions and the rule proposal at large may be worked out in larger detail, although restructuring supposedly takes place.

When using a consequent description in problem-maps, the same problem must probably be repeated several times (in various contexts). Since it is always preferable to reduce redundancy in legal as well as in other texts, the problem-map will be useful also when designing rule texts, especially with respect to a proposal about references.

It is interesting to note that while problem-maps (or problem sketches) are hierarchical, law texts etc. are mainly linear (except for references, indexes, cross-references by use of identical terms, etc).

One may assume that problem-sketches are used to some

extent in all preparatory law work. When writing out the text, a large degree of linearizing takes place. But when applying the law one needs, at least for the thought, a problem hierarchy in which to place the present problems.

If this hypothesis of mine about hierarchical ordering in the analysis phase, linear ordering as source, and hierarchical ordering again at the problem solving stage is near correct, then there should be a recognition of its usefulness in the classification of legal sources. Today this is exhibited down to the act level in the Norwegian Statute book's systematic register.

2 SPECIAL MODELS OF LEGAL PROBLEM SOLVING

In chapter 2 we shall introduce a notation for solution plans for legal problems. First, an abstract notation for problems is developed in paragraph 2.1, then the description of the problem is specified with respect to legal sources and facts in paragraph 2.2.

2.1 A general problem notation

The purpose of the notation is fourfold; it was originally introduced to cover two of these. There is a need for a bridge between the general model of legal problem solving and our special model (underlying SMARN), and for a clear demonstration of the limitations of SMARN.

Hopefully, the formalism is also useful when discussing other language alternatives. As such, a discussion of the formalism will in itself be of interest.

Often, one tries to make models for a small part of legal decisions, that is, for specialized areas. When making such models, one builds into them various assumptions. It may therefore be desirable to establish a frame of reference for the discussion of different models. Such a frame of reference is developed below.

2.1.1 From a general to a special model of legal problem solving

In chapter one we have discussed a general model of legal problem-solving. And we are looking for an implementable problem-solving model. May we perhaps implement the general model? In that case we have solved our task. But, unfortunately, the general model is far too little formalised to be implementable. This applies both to the identification

of problems and the solution of identified problems. Correspondingly, the problem-systematics and the solution-description are too little formalised.

Consequently, we need to establish a formal problem-solving model before it may be implemented. This formalisation will take place in the areas of problem-systematics and solution-description.

In this chapter we shall describe legal decisions solely in terms of these two processes: the identification and the solution process.

Let us start with the formalisation of the solution-description. The development from a general (GPS) to a special model of legal problem-solving (SPS), will be described by means of the two concepts problem-hierarchy and solution-plan (PHS). The chosen problem-systematics is in other words hierarchic. And solutions will be described in a plan.

PHS is a restricted version of GPS. The problem-hierarchy (see 1.7) is applied as a systematics for problems, while solution-plans (see 1.9) are used as solution descriptions. The solution-plans are assumed to be formalised in PHS, although no assumptions are made about the character of the formalism.

The steps of development will be

GPS ---> PHS ---> SPS

2.1.2 Problem - hierarchy and solution-plan

(1) Solving problems in PHS

We shall say that a basic problem, P, is any problem from the problem-hierarchy that may be solved. This requires that P=(x,F,R) is sufficiently specified, both with respect to conditions for its presence and with respect to solution-description when present.

The solution of a legal problem which has a solution plan is carried out by solving the parts of the problem according to the order prescribed by the solution-plan. This takes place for all the problems down to basic problems, which will be found in the problem-hierarchy. It is assumed that the solution-process does not modify the solution-plan.

The basic problems are then solved after a harmonization of the occurring norm-expressions attached to the actual problem.

(2) Limitations in PHS

PHS is a restricted form of GPS. We assume that the problems we will be asked to solve are either found in the problem-hierarchy or are defined by a solution-plan. PHS is mainly of interest as a bridge between GPS and SPS, rather than as a practical construction.

The interaction between the identification and solution of problems, and the feed-back possibilities from GPS, are stripped off PHS. This is a simplification. Generally, I will characterize the step from GPS to PHS as limitations of solution-descriptions for the decision-process.

2.1.3 Analogy between PHS and programming

We may compare basic problems in the problem-hierarchy with a high level instruction set for a computer.

If the legal sources, R, are collected, the problem, P, may be solved by interpreting R and applying them on the facts, F, of the problem.

Composite problems may be expressed by basic problems and other composite problems. They are expressed by solution-plans. Forming solution-plans may be compared with high level programming.

2.1.4 A special model of legal problem solving

SPS is like PHS with the exception that SPS contains a detailed description of the solution, L(P), for each problem, P. The difference consists in that there may be several legal sources for a basic problem in PHS, while there may be only one in SPS, i.e. a norm.

The solution-process in PHS for each basic problem P=(x,F,R) contains interpretation and harmonization of legal sources. In SPS, we assume that harmonization has taken place, and underlies the description of the solution for each problem. A variety of legal sources for a basic problem are thereby reduced to one source, making the problem-hierarchy superfluous.

The step from PHS to SPS may be regarded as a limitation in the description of basic problems - that is to say, limitations on the norm-level.

For PHS and SPS, identification of problems and establishing of solution-plans are not described in the model.

The solution-plan may therefore no longer be modified within the model as in GPS. The decision-process degenerates and becomes identical to the solution-process.

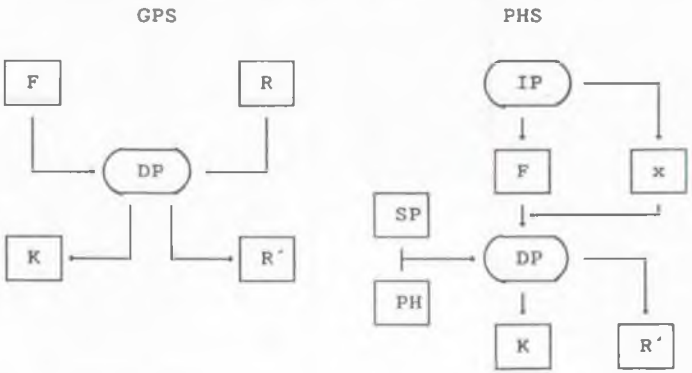


Figure 2-1: The decision-processes in GPS and PHS.

SP is solution-plan, PH problem-hierarchy, DP decision-process and IP identification-process.

For GPS it is necessary to underline the dynamic aspect of the decision-process. In the consequent, K, facts may be characterized differently from the way they are in F. The decison may also lead to a new legal source R´, and thus influence later decisions.

The same holds true for PHS. Here, the type-specification for a given problem, problem-hierarchy and solution-plan represent the relevant legal sources. The difference between GPS and PHS consists mainly in the fact that the solution-plan is defined outside the decision-process in PHS, and is therefore static within the decision-process. And when a problem is identified for a PHS-decision, it will not at a later stage be reidentified, as may be the case in GPS.

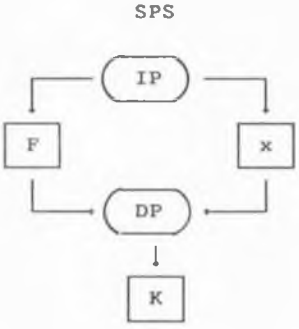


Figure 2-2: The decision-process in SPS.

In SPS the decision-process is described in detail. The legal sources get no contribution from the SPS solution-process.

SPS contains a description of the norms, so that to solve a given problem one needs only to interpret the description and apply it to facts.

In SPS one may solve only a finite number of problem-types. These are the problems that have a solution-plan and are defined in detail on a norm level. In the GPS and the PHS models, however, one may describe the solution of an infinite number of problem-types.

2.1.5 Well-formed solution-plans

To finish the chain of different problem-models we shall define well-formed solution-plans (WFS). WFS is a specific SPS.
A well-formed solution-plan may consist of the following elements and only these:

- the empty problem,

  ∅,

- a basic problem,

  n, (n is a well-formed rule)

  The semantics take the form: when solving the problem, apply rule n.

- a sequence of WFS´s,

  (P1, P2, ..., Pi),

  Semantics: When solving the problem, solve first P1, then solve P2, etc.; finally, solve Pi.

- alternative WFS´s,

  [P1, P2, ..., Pi],

  Semantics: To solve the problem, solve exactly one of the problems P1, .., Pi.

- repetitive WFS´s,

  P^*,

  Semantics: $P^* = (P,....,P)$, $P^0 = (\emptyset)$, $P^1 = (P)$,

2
P=(P,P).

- concurrent WFS´s,

{P1, P2, ..., Pi},

Semantics: To solve the problem, solve each of the i problems in parallel with no time restrictions.

- an arbitrary ordered subset of WFS´s,

<P1, P2, ..., Pi>.

Semantics: To solve the problem, choose any subset of problems from the set P1, .. Pi, order this resulting set in an arbitrary manner, then solve the problems in sequence according to this order. For example <P1,P2> is equal to

[ϕ, P1, P2, (P1,P2), (P2,P1)].

WFS´s may be named, for instance

P = (P1, P2, ... Pn)

gives the name P to the sequence (P1, .. ,Pn).

It should here be noted that the possibility of naming introduces problem definitions of the kind

P = (.... , P, ...),

etc. What should this be taken to mean? Precisely that when we solve a P-problem, then P itself is a partial problem. A practical legal example is to establish the group of persons who comprise the beneficiaries of a will. A sub-problem which frequently arises is that one must establish which group stands to benefit from the will of previously deceased offspring, who may themselves have living successors, etc.

These kinds of problems are called recursive.

2.1.6 Overview of the development of restrictions.

This overview reflects only problem-systematics and solution-description.

```
GPS  : (PS,SD)              PS = problem-systematics
                            SD = solution-description
  |
  |        PS --> PH        PH = problem-hierarchy
  |        SD --> SP        SP = solution-plan
  |
PHS  : (PH,SP)
  |
  |        PH --> PH'       one norm per basic problem
  |
SPS  : (PH',SP)
  |
  |      SP --> SP'         formal solution-plan
  |
     (PH',SP')
  |
  |      SP' --> SP''       basic problems into solution-
  |                         plan, making problem-hierarchy
  |                         redundant
     (ø,SP'')
  |
  |      SP'' --> SP'''     well-formed solution plan
  |
WFS  : (ø,SP''')
```

2.2 Problem notation with specified facts and legal sources

In this chapter we shall introduce more specific kinds of problems. This is necessary in order to describe a problem-solution in detail, since one needs a description of facts and sources.

First, we shall describe norm-expressions, which are legal sources.

2.2.1 Norms

I will not in this report define well-formed norm-expressions, but only discuss in some detail some of the more central, general aspects of norms.

Eckhoff and Sundby (Eckhoff:1975) regard it as useful to describe legal systems in terms of two types of components,

legal norms and activities. The legal activity we shall pay most attention to is legal problem-solving.

Alf Ross (Ross:1953, p.278) writes: All legal norms can be formulated: if x, then y.

The antecedent x specifies the conditions for the application of the norm. If the condition x can be said to be met, then the consequences in y shall follow.

In a particular case, x may either occur or not occur. Consequently, the discussion concerning whether the conditions are met has two possible outcomes: that is, the result of the discussion is binary. Compare this, however, with Gert Fredrik Malt's conception of probabilistic norms (Malt:1983).

The conditions in the antecedent may be composed of several sub-conditions. As for conditions as a whole, partial conditions may be said either to occur or not occur. Principally, each partial condition may be represented by a descriptive expression. The interpretation consists of assigning each expression a value true or false.

Partial conditions are composed into a whole by means of logical connectives: negation, conjunction and disjunction. Implication, equivalence, exclusive or and other connectives may be defined in terms of the basic operators.

The conditions may be expressed on legal relations, legal positions, actions, events etc.

Norms are divided into deontic and qualificational norms (Eckhoff:1975, p.66 and 84). This coincides with a partition of normative expressions into imperatives and qualifications (ibid, p. 84).

A deontic norm, n, may be described by four entitites

(O,H,P,B),

where O is a deontic modality, H is an action description, P is a reference to subjects influenced by the norm and B is a condition. The deontic modality may be permission, obligation,prohibition or exemption.

A deontic norm says that if conditions B occurs then P is

permitted or obliged to do the action H, or prohibited or exempted from doing it.

Let f1, ..., fn be n variables defined over domains d1,..,dn. Let F = (f1,..,fn) be defined over d1 x ... x dn. If fi is a legal variable, D may be regarded as a legal state-space. If each of the variables fi has been assigned a value, so also has the legal variable F. F is a legal state.

Mathematically speaking, norms map a legal state onto another,

n: D --> D.

A conditional norm has the form

if B then (O,P,H).

An example from the Norwegian penal code §271: Severe fraud is punished with a prison term of up to 6 years.

An unconditional norm has the form

if true then (O,P,H)

or equivalently (O,P,H).

A condition is a combination of criteria (atomic conditions) that in a given situation may happen to occur. Depending on which types of variable are used to describe facts, different types of atomic conditions will be defined.

Conditions are defined by means of operators and operands (partial conditions). As an example we may define well-formed conditions on a propositional level like this:

An atomic condition is an entity that is not described as a combination of other conditions and that has the value true or false. An atomic condition is a well-formed condition. If a and b are well-formed conditions, then

a and b,

a or b,
not a, and
(a)

are all well-formed conditions. The boolean operators not, and, or, have the corresponding priority. The parentheses have superior priority as operators. Hence,

a or b and c, is not equivalent to (a or b) and c

since when a is true and b and c are false, the former condition is true, while the latter is not.

Action themes prescribe the actions that can, shall, ought to .. take place if the condition of the norm is satisfied. The action theme will comprise a very heteronomous bunch of phenomena, such as delegation of power, how legal decisions are to be carried out, how organisations are to be established, sanctions, how valid acts are to be created, etc.

How powerful a formalism is, will to a large extent be determined by how varied the action repertoire is, and how powerful the action logic is.

By deontic modality one usually refers to permission, obligation, prohibition and exemption. One may think of these as deontic operators whose operands are actions. However, there exist a lot of different permission, prohibition, obligation and exemption concepts. Thus there is hardly any conceptualisation which is commonly agreed upon.

We may exhibit some of the relationships between the four concepts as follows, writing O for obligation, P for permission, F for prohibition and E for exemption:

P = not F, E = not O.

It follows that

F = not P, O = not E.

One usually lets the deontic operators act on states or actions. Let x be an arbitrary, well-defined action on which not x is also well-defined. We may then write

Ox for x is obligatory and
Fx for x is forbidden.

If we write shall for O and shall not for F, we have:
obligatory: shall x
prohibition: shall not x
permission: not shall not x
exemption: not shall x

Deontic logic contains inference rules for deontic expressions. The various analyses of deontic modalities yield various competing deontic logics, see e.g., (Hilpinen:1981), (McCarty:1985), (Castaneda:1981) and others. Operative systems should preferably use a logic which is paradox-free and with inferences comparable to the results of human reasoning.

Qualification norms may be said to establish relations between objects (x and y) in the form

x is qualified as y.

(Eckhoff:1975, p. 85)

Examples of different types of qualification norms may be conceptual definitions and qualifications of legal positions. Such a position may be "owner", for example.

An important group of qualification norms are those concerning competence. A common feature of these norms is that the competent agent has the power to declare norms directly, thereby influencing the competent agent himself (autonomous) or others (heteronomous competence).

In a poweful norm formalism, it is necessary to have constructions to express qualifications. These include conceptual structures, qualification of positions, as well as norms regarding competence. A fundamental work on competence, and on legal concepts as a whole, is Hohfeld's, in which he uses the four basic concepts competence, incompetence, immunity and submission (Hohfeld: 1923). See also (Pørn:1970).

2.2.2 Facts

A person's marital status, a person's age, the exercise of due care and the reasonableness of a price, may all be regarded as examples of facts. Let us imagine an atomic fact as a semantic entity describing a phenomenon. The specific occurrence of an atomic fact may take a certain value out of a set of candidates. Such semantic entities we shall call variables. The set of possible values is called the domain of the variable.

At present, no more will be said about which types of variables may be used. Sentence variables (where "sentence value" is synonymus with a sentence (occurrence)), phrase variables, (where a "phrase value" is synonymus with a phrase occurrence), word variables, (where a "word value" is synonymus with a word occurrence) are three possible types of variables from natural language.

How rich a formalism is needed to describe facts, depends on which variable types the formalism has, and to a large extent on what variable structures it has. The structure is important for getting efficient access to variables. This may be achieved by grouping together facts that are associated, by nesting variables so that, for instance, one may get part-whole relationships, by subordination and by use of multiple variables. Appropriate naming is also a prerequisite.

To describe facts in a given problem domain certain fact types and structures are needed. In choosing a representation, one must ensure that it is powerful and efficient enough for the purpose.

An important aspect concerns how much semantics one wishes to associate with the primitives and the structure. If one operates with a part relation, it follows by its transitivity property that if a partof b and b partof c one may deduce a partof c.

If one operates with the taxonomy relation and it is given that a speciesof b, then the properties that a has, are also properties of b.

Such inference rules may be associated with the structure primitives. And to administer these rules one needs an inference mechanism.

The first question to be raised is whether one is to choose a knowledge representation or a data representation of the facts. To decide, one has to know how the facts will be used. For reasoning with alternatives, one chooses knowledge representation; for calculations involved in making plans one chooses a data representation.

Another point concerns how many primitives we will have in our data representation: time, events, situations, actions, roles, sets, bags, records, sequences, arrays, etc. We will not decide upon the type primitive question, only the structural primitive question. And the goal is clear: the fewer the better, and the more powerful the better.

It is neccesary to be able to associate variables (attributes) together into records (objects) and give them names. This is especially important if one needs a large number of occurrences of equal objects. As primitives for multiple occurrences one can think of bag, set, array, sequence, and others. We shall choose sequence. It may be used to implement both bag and array, but not set if one allows equal object types in sequences.

Associated variables are also important if one requires complex structured variable types. If one is to describe a part relationship efficiently, one must be able to express the idea that an object is a part of another, as, for example, when we speak about the dynamo in a car.

Records will be useful since the same part may occur in various connections. And each part is declared only once.

The association of variables will be horizontal (side-ordered variables) or vertical (subordination).

We shall now introduce a notation for the description of structured variables, named well-formed variables. The purpose is to establish a general framework for fact descriptions.The name may sound strange. The reason for this is probably that the idea of structured variables is not too frequently encountered.

An atomic variable is a variable of one of the basic types. An atomic variable, i, of the basic type integer is declared

integer i

in many programming languages. An atomic variable is a well-formed variable. In BNF (Backus-Naur Form) we may define an atomic variable:

<atomic variable>::= variable <basic type> <list of variables>.

The line above is read: an element of the syntactic category called "atomic variable" is defined as an element of the syntactic category "basic type" followed by an element of the syntactic category "list of variables".

Structured variables are of two kinds, records and sequences: Records are declared in the form

variable (<recordtype>) <list of variables>

<recordtype> is a list of variables separated by "," or " ". If a and b are variables, then a,b shall mean that a and b are equal in rank, while a b shall mean that a is superior to b.

variable (a,b) c is a declaration of a variable c, consisting of an a and a b variable.

Sequences are declared in the form

variable <range> <<recordtype>> <list of variables>

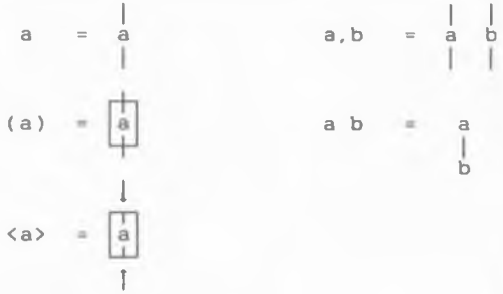
where

<range>::= <lower limit>:<upper limit>/<length>/o

Here, / represents alternatives. In <<recordtype>> the angle brackets have two different meanings. The inner pair signals a syntactic category. The outer <,> are used to signal sequences. (,) are used for records. <length> is an integer expression to indicate the length of the sequence. <lower limit> and <upper limit> are integer expressions defining lower and upper limit for the index of the sequence.

Structured variables are well-formed.

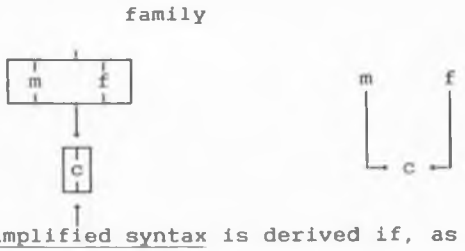
Since a visual presentation is often more compact and easier to read than a strictly textual presentation, we shall define a graphical syntax for well-formed variables:



According to these rules

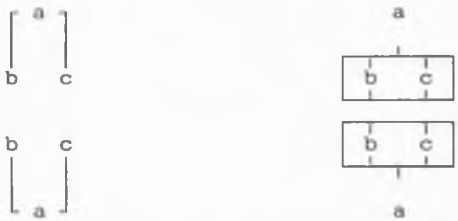
var ((mother,father)<child>)family

may be represented



A simplified syntax is derived if, as a governing rule, one does not draw the boxes and the loose ends, but only the branches (possibly directed) between the atomic variables, as is done on the right.

One may transform a representation in the simplifyed syntax to the complete form by applying the following rules:



Corresponding rules are valid if we substitute a — b with a ⟶ b, a ⟵ b, etc. However,

are not expressible in the complete syntax. There is still a problem to be solved concerning the number of boxes to be drawn around variables. This is solved by a convention that default is the least possible number of boxes. While any additional box is represented in the simplified syntax as well.

Example 1:

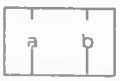
If a and b are variables then

(a,b), (a b), <a,b> and <a b>

are types of variables there may be declared variables by,

e.g.

variable (a,b) v;



Note that v is the variable to be declared, not a sub-ordinate of (a,b).

Example 2: Arrays

variable 100<a> b; is a variable consisting of 100 a-elements, named b.

variable n<a> b; has n a-elements.

variable m<n<a>> b is a two-dimensional variable of type a with m rows and n columns.

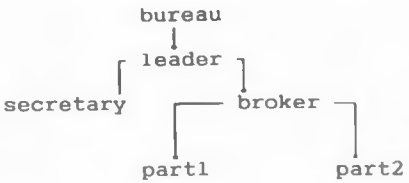
variable m<<a>> b will have m rows and an undefined number of entries for each row.

Example 3: Tree-structure

Suppose that we wish to describe information for a company of brokers organised in separate bureaus, each having a leader, a secretary and one or more brokers, and each having cases with two parties to each case.

var <leader(secretaty,<broker<part1,part2>>)>bureau

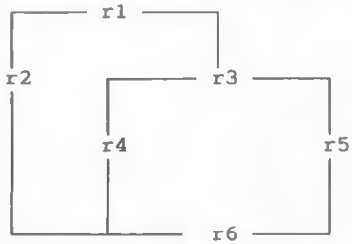
now represents the whole structure:



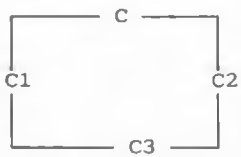
Here, a — b represents a 1-1 relation while a ⟶ b represents a 1-n relation between a and b.

Example 4: Hierarchical structure

variable (r1 (r2,r3 (r4,r5))r6) b may be visualized



variable (C<<(C1,C2) C3>> b may be visualized g;



The mechanisms for declaring variables are not strong enough to catch all kinds of hierarchies as structured variables without using reference variables. The simplest example is the following, expressed in simplified syntax:

On the right, the conflicting requirement is stated in the complete syntax.

Example 5: Network

An example of a many-many relation with subordination:

variable (<leader> <member>) projects

visualized:

An example of a many-many relation without subordination is:

variable (<buyer>,<seller>) market

There may be different relationships between the seller and the buyer represented, for instance by references.

2.2.3 Solution-plans in (F,R)-notation

In (F,R), we may think of F as a fact structure (or variable structure) and R as a legal source structure. R may refer to and change values of the variables in F. (F,R) is an association between a fact structure and a legal source structure. This association constitutes the solution of a problem. F also gives R's variable context.

In our attempt to automate the solution of legal problems, we have now developed a well-formed solution-plan. This is expressed by problem identifiers P, P1, P32, etc., structure symbols in the form of parentheses (, [, <, { and the repetition symbol *. It is now time to design well-formed solution-plans for problems described by facts and norms. For no problems may be solved without details of facts and norms.

Would it now be correct to make substitutions of the form

P ::= (F,R)

in well-formed solution-plans, WFS(P), to produce well-formed solution-plans for (F,R) specified problems, WFS(F,R)? Yes, this would give us a class of languages capable of solving problems. What these languages would look like would vary according to which variable types and legal sources they contained.

But since we say in 2.2.6 that the solution of (F,R), L(F,R), shall operate only on F,

L(F,R) =R(F),

then languages of the type WFS(F,R) would be very cumbersome to use. This is especially the case with respect to dependencies between problems. If, for instance, P1 = (F1,R1) and P2 =(F2,R2), and we have a problem P = (P1,P2) to solve, then by substitution we have

P = ((F1,R1),(F2,R2)) and
L(P) = (R1(F1),R2(F2)).

If P2 depends on P1, this may only be expressed by letting F1 and F2 have common elements. Therefore, one must state, for each problem, which facts it is to operate on. This could be done as follows:

1) Global declarations (correspond to a definition of a fact type) of all variables to be used,

2) Local specifications of all variables to be used locally.

(Dijkstra :1976 p. 83) has proposed a reference model of this sort. This is a neat solution, especially with respect to verification.

We shall not use this mode of reference in the (F,R) formalisms, but one corresponding to that used in Algol and SIMULA.

In addition to substitutions like P ::= (F,R), substitutions like P ::= R and R ::= (F,R) will be allowed.

Thereby, one may introduce new problems without introducing new variables (P ::= R). And one may use structured legal sources (R ::= (F,R)).

Returning to the reference rules, let us introduce the following rule: A legal source may refer to variables in the same or outer parentheses. It may not refer to variables on deeper parenthetical levels.

What motivates for such a rule of reference is, among other considerations, that legal sources are closely related to the facts they operate on, that one can more easily verify that a solution is correct, that one may allow different uses of names and elegantly resolve naming conflicts by applying the rule of 'closest definition', and that fact types are not introduced before they are needed.

Let us now summarize the syntax involved in the production of WFS(F,R):

WFS(P): L ::= L,P/P/P*
P ::= ø/N/(L)/<L>/[L]/I/{L}/ I=P

In the syntax for well-formed solution-plans for P-specified problems, L is a list of problems, ø is the empty problem, N is a basic problem and I a problem identifier.

WFV(F): F ::= f/(T)/<T>/<range><T>
T ::= T,F/F/F F

<range> is defined in 2.2.2. In the syntax for well-formed variables F is a structured variable and T is a tuple of variables.

WFR(R): N ::= R

Together with WFS(P), this is the syntax for well-formed legal sources.

SR: N ::= (T,L)/R

This is the syntax which permits the nesting of legal sources.

2.2.4 How to refer to variables

The purpose of naming phenomena is to refer to them.

For atomic variables, we are also interested in referring to their values. If we are interested in assigning a value to a variable, then we are primarily concerned with identifying it. Otherwise, we are mainly interested in its value. In programming languages, information as to whether a reference to a variable identifies the variable or its value, is usually carried implicitly.

```
y:=y+3;
```

is a valid statement in many languages, where the y on the left-hand side of the assignment operator (:=) refers to the variable y, while the right-hand side y refers to the value of y. Thus the effect of the statement is to increase the value of the variable y by 3.

On the basis of the variable structure already specified, some modes of reference may be introduced. Further modes of referring cannot be described until basic types have been introduced.

A legal source refers to an atomic variable by its name. If we want to speak about the age of a person, for instance, age may be introduced as a variable.

If p is a reference to a person-record, containing among other things a variable age, then 'p's age' shall be understood as a reference to the age of p. We adopt SIMULA's notation p.age for this case.

If person is a reference to a sequence of persons, then person(i) will be a reference to the i-th of these, i being an integer, obviously.

If leader is a reference to a leader of a firm with

several employed persons,

```
var (leader <employed>)firm;
```

then

```
leader"employed(i)
```

shall be a reference to the i-th of these. a"b shall also be read 'a's b', and is used to refer to a's subordinate variable b. But isn't " superfluous? Couldn't we write a.b for that case as well? No, not if we want to allow equal variable names locally in different variables. If we have

```
var<a (b,c)>d;
var(b,e)a;
```

what then is d(i).a.b to mean?

2.2.5 Solving (F,R)-specified problems

We shall now discuss the solution of (F,R)-specified problems, that is, problems in which the legal sources and fact-structures are given. The solution of the problem (F,R) may be designated by L(F,R). In general, we may have

L(F,R) = (F',R'), or L : (F,R) --> (F',R'),

where R'is not necessarily equal to R, nor F' to F. This means that both the fact types and the legal sources may be changed by decisions L.

2.2.6 Solutions shall not modify legal sources and fact types

We restrict our discussion to the case in which

L(F,R) = (F,R),

that is, where the involved legal sources and fact types are not amended during, or due to, the decision. Only the facts' values may be changed. Therefore, we will also write

L(F,R) = R(F).

The reason for this is technical: in order to be able to compile code it is necessary that the code is not self-modifying. And we want to be able to compile our code.

2.2.7 Describing solution-processes in WFS(F,R)

We have defined well-formed solution-plans and variables. Here, we shall describe the solution-process in some detail.

The evaluation rules are illustrated by the following example. e(R) represents the evaluation of R, etc. If R = (R1,R2) then e(R) = (e(R1),e(R2)). The same holds true for concurrence, alternatives and arbitrary subset. If R = R1*, then e(R) = (e(R1))*.

For (F,R), we have e(F,R) = (F,e(R)).

To a given basic problem, or legal source, R, there will correspond a set of relevant facts,F. Assuming that for each legal source we know the corresponding relevant facts of a case, the solution cyclus for a problem P=(R,F), consists of the following steps:

- interpret R,
- evaluate F,
- apply R to F.

In a prefix notation, we may write

L(F,R) = Apply(Interpret(R), Evaluate(F))

(1) Interpretation

We assume that the norms are harmonized. Therefore, we shall comment on two issues only. The first is that the interpretation leads to an "understanding" of which facts are needed to apply the norm to the actual case. We may make a distinction between input and output facts for a given norm. Before the evaluaton of a norm, one needs to know the input facts only, while output facts are the results of the application of the norm.

Here we may mention the obvious point that the interpreter of the norm must know where and how the facts are to be supplied. The interpretation also leads to an "understanding" of which operation to carry out.

(2) Evaluating facts

When interpreting a legal source, a kind of model of the relevant facts is established or activated. This model must get its values in the specific case. If a value of a fact is found in the expected location, only an evaluation of it is needed before the value is ready for use. (Here, proof-procedure and reliability-evaluation come into play in the traditional legal systems). But if the fact is not found, it must first be supplied before it can be evaluated. Graphically, the evaluation of facts may be described:

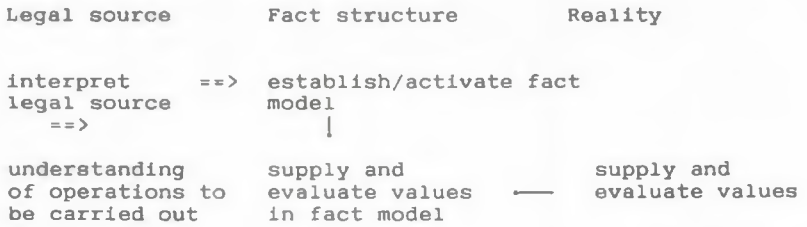


Figure 2-3: Interpreting legal sources and evaluating facts.

a ===> b represents a has the effect b. a ---> b represents a is a prerequisite for b

2.2.8 An example of a language

Let us now introduce a language in order just to illustrate some of WFS(F,R)'s possibilities. Let us operate with the basic types integer, boolean and reference for variables; and let us suppose we want to use the following types of legal sources:

```
if B then P1 else P2      (if-statement)
V:=E                      (assignment statement)
while B do P              (while statement)
```

B is a boolean expression, V a variable and E an expression. How can we use WFS(F,R) to implement such a language?

We know that the if-statement is a kind of alternative statement and that the while statement is a repetition statement. Let us therefore check whether the if and while statements may be expressed as interpretations of the WFS(F,R) constructions.

But first a more complete version of the grammar, leaving the expressions B and E and the variables undefined:

```
R ::= C/S/W
C ::= if B then X else P
S ::= V:=E
W ::= while B do P
X ::= S/W
```

Let B have boolean operators to combine boolean expressions and comparators to compare, respectively, arithmetic expressions and reference expressions. Let E be arithmetic, boolean or reference expressions.

Alternatives can, indeed, be used to implement if-then-else. Before we give the details, we shall introduce a specified alternative notation:

```
LTR[B1 -> X1, B2 -> X2, .. , Bn -> Xn].
```

Here, B1 -> X1 shall mean if B1 then X1. LTR (left to right) means that the B´s are tested from left to right.

The first, and only the first, condition found true will be selected, the effect being that the corresponding rule is activated. No error occurs if no B´s are found true.

[B1 -> X1, ... , Bn -> Xn]

corresponds to Dijkstras non-deterministic if-construction (Dijkstra: 1976, p.33), except that a fault situation does not occur if no B´s are found true. But note that the definiton of alternative is a special case of the (F,R) notation´s alternative construct. The if-then-else statement may now be implemented by

LTR[...].

if B then X else P = LTR[B -> X, not B -> P],
= LTR´[B -> X, P].

In the last alternative, LTR is modified in such a way that if the last condition is met, which can be true only if all the previous conditions were false, it is asserted to be true, and can therefore be omitted.

So by replacing C by C´

C´ ::= B -> X

and interpreting [,] as LTR´[,], we have introduced if-then-else as a specific form of alternative construct.

while B do P

may be implemented by

((LTR´[B -> P, goto out])* ,out:).

So by replacing W by W´,

W´ ::= goto I,

interpreting [,] as LTR´[,] and introducing labels

D ::= I:/Dl:,

and replacing R by R´, where

R´ ::= DC´/DS/DW´/C´/S/W´,

one has implemented the while statement in WFS(F,R).

A program example:

Let us say there are a hundred persons in a register. We would like to find the average age for the ones who are married. For this task we need to know if a person is married and his age. If we assume var integer age and var boolean married, then we may declare structured var 1:100<age,married>person

There is also a need for var integer sum,no,ave,i to tell the sum of ages, number of married, average age and number processed.

Our simple program will then be:

```
(var integer sum,no,ave,i,
    (var integer age, var boolean married,
     var 1:100<age,married>person,
     i:=1, while i ≤ 100 do
     if person(i).married then
         (sum:=sum+age,i:=i+1,no:=no+1)
     if no > 0 then ave:=sum/no))
```

2.2.9 SIMULA in (F,R)-notation

Let us now try to describe a programming language in the (F,R)-notation. Let us take a look at SIMULA. The review is intended merely to illustrate more specific interpretations of

a (F,R)-notation.

(1) Basic types

SIMULA's basic types are boolean, integer, real, longinteger, longreal, character, text and reference. In addition SIMULA has a data-structure primitive, namely array. An integer array i with 100 elements is declared integer array i(1:100) with similar meaning to var 1:100<integer>i in well-formed variable notation. A syntactically correct SIMULA program will have variable declarations transformable into well-formed variables.

(2) Sequence of imperatives

Statement sequences are made in SIMULA by putting statements after each other, separated by semi-colons.

```
    st1;st2; ... ;stn
```

is an implementation of

```
    (st1,st2, ... , stn)
```

where the st's are statements. The word rule is used for statements in the (F,R) notation.

(3) Alternatives

For the if-then-else construct, see 2.2.8.

Similarly,

```
    switch l :=l1,l2,s(m);
    switch s:= s1,s2;

   goto l(i);
```

may be implemented by

```
LTR[i = 1 -> goto l1, i = 2 -> goto l2, i = 3 ->
     LTR[m=1 -> goto s1, m=2 -> goto s2]].
```

(4) Iteration

I shall next illustrate some of the iteration possibilities in SIMULA.

Example 1:

```
    for i:=1 step s until u do st;
```

may be implemented by

```
    (i:=1,(LTR´[i ≤ u -> st, goto out])*,out:).
```

Example 2:

For the while statement, see 2.2.8.

Example 3:

```
    for i:=e while b do st;
```

may be implemented by

```
    (i:=e;(LTR´[b -> st, goto out])*,out:).
```

(5) Block

SIMULA has three implementations of problems, P. There are blocks, procedures and classes, respectively. A block may contain declarations (of variables, procedures, classes, etc.) and statements.

Let T = Tg U Tl. U here means union. Tg are textually surrounding, or global, declarations of variables, procedures, classes, arrays, labels and switches. In SIMULA global variables may be referred to directly. Tl is a local declaration of variables, procedures, classes, etc.

```
(T,L) = (Tg U Tl, L)

is implemented

begin
    decl Tl;        (decl Tl is an abbreviation for
    L;               declarations of variables Fl).
end;
```

The context of the block, i.e. the environment L will be evaluated in, is Tg. By interpreting (T,L) as above, one may implement blocks. In "An essay of the notion: The scope of variables" (Dijkstra:1976, p.79-93) proposes the use of specifiers for blocks: this could be an improvement for several reasons, among these ease of verification.

In SIMULA the following substitution is allowed:

```
    R ::= (T,L).
```

R is a legal source. This means that wherever R may occur in a SIMULA program, one may create another SIMULA program by substituting R with a block (T,L).

If

```
begin integer a;a:=25;R1;end;

is declared and R1 is

begin integer b;b:=25;a:=a*b;end;

then

begin
    integer a;
    a:=25;
    begin
        integer b;
```

```
        b:=25;
        a:=a*b;
    end; end;
```

is a syntactically correct program, since the two parts above are correct and the second part has been substituted for R1.

(6) Procedure

Let T = Tg U Tp. Tg are global declarations and Tp formal parameters. For SIMULA, we need an implementation of (x,T,L) where

```
(x,T,L) = procedure x(Tp); spec Tp;L;
```

spec Tp refers to a specification of the formal parameters required by SIMULA. When x is executed, it will take place in the environment (Tg U Ta), where Ta are the actual parameters for Tp, defined in the call

```
x(Ta);
```

SIMULA allows constants, variables, expressions, arrays, labels, switches and procedures as parameters to procedures. These are transferred in three different ways, by name, by value or by reference. In call-by-value, the actual parameter is evaluated and stored in a local variable that the formal parameter refers to.

For call-by-reference, a local copy of the reference variable is established. This is initiated to the actual parameter by the call. As a result, the formal and actual parameters point to the same object immediately after the call. Arrays, procedures, labels and switches are transferred by reference, if nothing else is specified.

Call-by-name represents a textual substitution (replacement) of formal parameters with actual parameters.

(7) Class

Let T = Tg U Tp. In SIMULA one may declare a class of objects which may contain both variables and statements.

class x(Tp);spec Tp;L;

declares a class x, with formal parameters Tp and body L. The class above will be regarded as an implementation of

<(x,T,L)>.

Syntactically, this looks like a (variable) sequence of procedures (x,T,L) with formal parameters Tp and body L. And, in fact, there is little wrong with such an interpretation.

A class's attributes consist of formal parameters, entities declared in what is called the virtual part and declarations on the outermost block-level of the class.

In SIMULA one refers to attributes by expressions of the form

x1.a,

where x1 is a reference to an object, say of the class x, and a is one of its attributes. If such a reference to an object exists, one may thus change attributes' values if they are variables or arrays, or call the evaluation of procedure attributes.

Classes may only have variables and arrays as formal parameters. Classes may not contain declarations of classes. <(x,T,L)>, where T contains <(y,T2,L2)>, is consequently not allowed in SIMULA.

(8) Subclass

Let a be a class-name.

<a (x,T,L)>

may be interpreted

a class x(Tp);spec Tp;L;

We have hereby imitated SIMULA's way of declaring subclasses. x is a subclass of a.

(9) Some speculations about variable subordinations for variables, classes and procedures.

For the sake of curiosity, let us consider various ways of interpreting variable subordinations. Obviously, this is also applicable to SIMULA. The constructs are not implemented in SIMULA, however.

Example 1:

T1 (x,T,L)

may be interpreted as a complex-type procedure:

var F1 procedure x ...;

Correspondingly for T1 T2 (x,T,L):

var (T1 T2) procedure x...;

Example 2:

(x,T,L) T1

may be interpreted as subordinated variables T1 to the procedure x. For example, T1 could be referable as long as x is. Thus T1 may be referable after x is evaluated.

Example 3:

T1 <(x,T,L)>

could be implemented as a form of type-class

    var T1 class x ... ;

The meaning here could be that results from the evaluation of objects of the class x are stored in T1.

Example 4:

    <T1 (x,T,L)>

would be analogous, but with one variable for each object instead of one for all. Retaining a notation analogous to that for subclasses, <T1 (x,T,L)> should be read

    var T1 class x ...;

in conflict with example 3. This can be resolved by inspecting T1's type, variable or class.

In this way, one may speculate rather freely on combinations of variables, classes and procedures. But hopefully, the above is sufficient to illustrate an interesting aspect of the notion introduced , namely the idea-generative aspect.

(10) Record

There are no constructions in SIMULA corresponding to what we call a record:

    (f1,f2, ..., fn).

The closest will be a class that may have only one occurence and where the class's only contents are attributes on the outermost level. Therefore, there is no natural interpretation of (f1, ... ,fn) in SIMULA.

    There are reasons, however, for introducing a record concept. The most important one is simplicity of

declaration of static variable structures. Various statements could be defined for complex variables, etc.

(11) Rules of substitution for SIMULA

As has been indicated, there are three especially important rules for substitutions in SIMULA. These are

1) R ::= (T,L), (blocks)

2) F ::= (x,T,L) and (procedures)

3) F ::= <(x,T,L)>. (classes)

3 follows from 2 and the structured variable rule, saying that if F is a well-formed variable, then <F> is also a well-formed variable. (I am stretching the variable concept rather far here. In ordinary jargon it should speak of declarations instead of procedure variables. But since I like the idea of procedure variables, which is not my own, I let this unorthodoxy stand).

2.2.10 Operations on structured variables

Sometimes it is useful to describe transformations on variable structures. Here I will just mention three of the more important operations:

(1) Exclusive choice

If v1,v2, ..., vm are variables, then the exclusive choice of one of them is also a variable.

(2) Selection of a record

If <v> is a sequence of v-records, then the selection of one of these would give us a specific record in the sequence.

(3) Projection of a sequence

If <v> is a sequence of v-records, then a projection of each record gives us a sequence of projected variables v.

3 DESCRIPTION OF A SPECIAL MODEL FOR LEGAL PROBLEM SOLVING, SMARN1

So far, a general model of legal problem-solving has been described (in chapter 1), and an abstract formal notation for solutions of problems has been introduced (in chapter 2). Therefore, the time is ripe to start the discussion of the specific design steps taken in SMARN. SMARN1 refers to the original version, which reflects the syntax described in working document SI (Hansen 1982).

SMARN2 refers to the version designed to give a satisfactory description of the solution of our first practical problem, a class of problems concerning inheritance.

3.1 Norm-theoretic elements in SMARN

We assume in SMARN1 that the subject reference is given by the problem. Therefore, we consider the action theme and the deontic modality only.

3.1.1 Rules and guidelines

Norms may be separated into two exhaustive groups, rules and guidelines. While consequences follow immediately when a condition in a rule is satisfied, this is not the case for guidelines. The guidelines are attached to weighings. For these it is typical that no one factor alone may determine the result. On the contrary, one has to weigh the different relevant factors against each other before the result may be determined.

Typical guidelines concern which factors are relevant in a

weighing, in what direction the factors pull and which weights they are to be given. Judgements concerning relevant factors and weighings may be particularly discretionary, and thus leave open the nature of the result.

3.1.2 Norm structure

There are a number of different structures holding between norms. Different norms may refer to the same word. Such common words may, for example, be established by legal definitions or by what are called weighing markers. These are words signalling that a discretionary decision is to be made. The names of legal relations or positions may occur in the consequent of qualification norms or in the antecedent of deontic norms. Eckhoff and Sundby designate as static such structures as those mentioned above.

They partition dynamic structures into genetic and operative connections. Genetic connections are connections between a norm and the norm that granted the competence required for its generation.

We will not introduce genetic connections into our models (in contrast to what is planned in LEGOL).

By operative connections Eckhoff and Sundby mean that succesive stages in a series of events are regulated so that the norm(s) which may be applied in one stage is/are decisive in regard to which norm(s) may come into action in the next stage.

We shall certainly take care of operative connections in SMARN.

3.2 SMARN1 limitations

In SMARN1 it is assumed that legal problems may be solved hierarchically. This is a reasonable assumption if the same problem is not solved several times in the same case. This restriction is removed in SMARN2. Here, we may operate with a recursive problem structure.

> The concept of solution is used with different meanings. Here, I shall take the opportunity of mentioning three different meanings: In the sense of a solution-prescription expressed by norm-expressions; in the sense of a solution-process; and, finally, in the sense of the result of a problem.

3.3 Facts

Thus far, our model of a legal problem looks like this:

P = (x,T,L),

where x is a problem-denotation, T are the relevant fact-types for the problem and L is a sequence of norms (cf. 2.2.3). We imagine that the solution of P is described in a norm-segment.

The fact-model needs some elaboration. In SMARN1 T is a list of types of fact,

T = F1,F2, .. ,Fm.

Each fact represents a type of action, event, situation, property of a legal object, legal connection, or several of these. In the last case, for instance, we may imagine that several facts represent an action.

This fact-model is poorer in structure than the well-formed variables described in chapter 2. Neither the record, nor the sequence concepts are implemented. But facts collected in one scheme will correspond to some degree to the record concept.

> "Facts" will here be used in the sense of "legal facts" when they may not be regarded as general facts. "Legal facts are the facts that immediately bring a rule to act". (Bratholm:p.329). Proof facts will not be considered in SMARN. We restrict the use of the model to problems where the legal facts are quantifiable.

At the outset, we assume that the fact-types are represented by a descriptive expression. As an example:

Dx25 = the client's age is 25 years

To model ages from 0 to 110 we need 111 fact-types. This is very tedious, so we shall allow the expressions to contain variables. Our expression may be restated as

The client's age is x years,

where x is a variable having as its domain the integers between 0 and 110.

In a specific case the actual value for x may be 25. It will generally be the case that the variables referred to in expressions for fact-types have their values set (to constants) in particular instances.

What is it in the first case that may be quantified? Obviously, the situation is that for

Dx0, ..., Dx110

only one of the expressions may be true. The truth-value is the value the expressions are there assigned. This could be represented as a variable for the fact as follows:

Dx25 = it is x that the client's age is 25 years.

x is a variable with domain true and false.

In SMARN, four basic types of variables are used: real, integer, boolean and text. An example of a text may be names:

Dx1 = The client's name is x.

In a particular case x may be Peter Smith.

We may sum up by saying that each fact-type, (Dxj,Fxj), is represented by a descriptive expression and an associated variable, Fxj. This will have its value assigned in any specific case.

The descriptions will not be an integral part of SMARN, but may be included in the comments to programs.

There are severe limitations in the way we represent facts: We lack an advanced representation of concepts, as conceptual hierarchies or semantical nets. Consequently, we are unable to represent one fact as a species of another, e.g. that a carrot

is a vegetable. This is often useful, but the restriction fits a "schema as input"-delimitation which the project has adopted.

Within the AI field, there is a good deal of work on conceptual hierarchies. McCarty, among others, has done very promising work relating to legal applications.

SMARN1 lacks tools for expressing complicated connections. If we wanted to model the fact that A owes B $10, we would have to establish the fact-type (D,F) = A owe B x $, where x is a variable and A and B unknown constants. A and B are thought of as references to two specific persons in the case. But what if we needed to represent an owing-relation for a group, in order to solve a problem? What should we do then? It is not possible to make A and B variables since we allow only one variable in each expression. One solution would be to introduce persons C and D, etc, and corresponding fact-types. But this would soon prove to be an impractical solution. In 4.8 we shall return to the discussion of mechanisms for handling relations.

There are many fact-models capable of handling complex relations. Relational databases (Date:1981), deep-structural models (Nijssen:1981), the entity-relationship model (Chen:1979) and semantical networks are implementations of some of these.

Fuzzy logic may be an alternative to propositional logic for describing facts. For some facts, it is obviously more appropriate to say that they occur to some degree in a case (which is easily expressible by means of fuzzy sets) than to say that they either occur or do not occur.

It would complicate the project to use fuzzy logic. Few programming languages support it, while propositional logic is extensively employed.

Before leaving the discussion about the representation of facts, I want to stress that up till now the fact-model has been described only for a partial problem. If we consider the fact-model for the superior problem together with its partial problem, the fact-model is definitely more powerful.

3.3.1 Variable declarations

(1) Case, norm, temporary, accum and result variables

case <type>, norm <type>, temporary <type>, accum <type> and result <type> are the different variable specifications in SMARN. These give the information the preprocessor needs for the secondary storage specifications it must generate.

case-variables are case-specific, norm-variables norm-specific (variables associated to norms), temporary variables exist only during calculations, accum-variables are used to accumulate values for statistical calculations for several problems and enable the end-user to refer to their value, while the result-specification is used to signal storing after the execution of programs has finished.

accum-variables may only be declared globally.

case, norm, and result variables must have a specification of their length, for example

```
case real r(5,2);
case integer i(6);
case text t(8);
```

This information is used for formatting on external storage and VDU, for example

```
case boolean x;
```

is now a definition of a case-specific, boolean variable x.

(2) Stochastic consequent

Some rules allow persons to choose between a set of action alternatives. SMARN has a way of handling this for statistical purposes. We introduce a concept of stochastic consequence. It involves nothing more than associating an expression of relative frequency with each of the different alternatives.

A stochastic consequent has the syntax

<stochastic consequent declaration> ::= stoc <name> = <sa list>

<sa list> ::= <sa> / <sa>,<sa list>

<sa> ::= (K,<arithmetic expression>)

K is a consequent.

Example:

Let a, b and c be consequents and i1, i2 and i3 relative frequencies for a, b and c. We may declare a stochastic consequent

stoc x = (a,i1),(b,i2),(c,i3);

A stochastic consequent may be activated from a statement list as in

if d then x;

Here, x will be evaluated if d holds true, the effect being that a, b and c will be evaluated, one after the other.

(3) Punk specification

Each fact (factor for example) for which one wishes, interchangably, either to give a value as input or to have a type-procedure to calculate, must be specified as a punk variable. Punk is an abbreviation for probably unknown value. A punk variable x, is specified

punk x;

(4) Population size

A user may do calculations for various population sizes set by himself. For each case he may set a size. POP = 1000 defines a

population of size 1000, for instance.

If one uses stochastical consequents on different levels, one must operate with actual population size. If citizens may choose between alternatives a (20%), b (30%) and c (50%), then the actual population in a is 200, in b 300 and in c 500. Further, if there is, in c, a stochastic consequent with alternatives x (10%), y (10%) and z (60%), then the actual population in x and y is 50, while in z it is 300.

One may refer to the population sizes via SMARN variables named

pop - total population, and

cpop - current population. Consequently, these variable names may not be declared in a SMARN program.

3.4 Rules

We shall describe a form in which to represent a norm, R, which is more compact than the simple if-then form:

```
R ::= K /
      if B then K /
      if B then K else A

A ::= K / R
```

B is a condition, K a consequent and A an alternative consequent.

A norm will be regarded as a solution-prescription for a problem in our system. This is the only function we will aim to capture by means of norms. That this is a limitation should be rather obvious, bearing in mind the effect of norms on the citizens' behaviour. Imagine that the problem we face is to predict the economic effect of a planned act on a population. At the outset, we may assume that the different norms influence the various persons' behaviour differently. This can be reflected only by a change of behaviour parameters, since we do not aim at developing mechanisms for the description of

dependence between norms and behaviour.

3.4.1 Operational norm

We may define an operational norm as a norm for expressing an operational connection. An operational norm might be expressed in the form

<apply statement> ::= apply <apply rule>

An <apply rule> is a rule where the consequent is just a number, referring to a given weighing function.

It seems that such norms more often occur in the more specific form:

    This <section> contain <x-problems>,

where, immediately after, the <section> follows. Rewritten, this reads

    apply if <x-problem> then y;
    "y"

Here the norm may be regarded as a port to y. But a jump, so to speak, is most probably rare:

    apply if b then n;
    ...
n: " the n- regulation".

This is the reference form, e.g. "Cf. section 8.2", etc. These references may be made operational by prescribing analogous solutions.

To form an explicit description of a solution of a problem, the operational structure must be described explicitly. This may be done in two ways: Either by extending the conditions for describing operational requirements, or by putting operational information into the

consequents. I will propose the latter solution as a standard, because the first one will lead to very complex antecedents in deeply nested norm-structures.

3.4.2 Conditions

The conditions, B, in a norm define the circumstances under which the norm may be applied. The conditions may be expressed by the variables.

The client´s age > 17 years

is an example of a condition. If x represents the client´s age, the condition may be expressed

x > 17.

Conditions may be expressed in different logics. Two or multivalued, Boolean, first order, second order logics and others.

Conditions in SMARN1 will build upon Boolean logic. This decision is not a matter of principle, but a practical solution, since <Boolean expressions> is handled efficiently in the language SMARN is to be installed in, SIMULA.

A condition is a descriptive expression that may be assigned the value false or true. An atomic condition is a condition that is not described by a combination of partial conditions. Partial conditions may be constructed using the logical operators and, or and not.

Comparators may be used to establish atomic conditions. These may be used with real numbers, integers and texts. The operands may be arithmetical expressions, variables or constants.

<, $\leq$, =, <>, $\geq$, and > are the 6 standard comparators for arithmetical expressions.

x+y > 100 000

for example, is a condition where x and y are two variables that together with the plus sign form an arithmetical expression. The condition requires that the value of this expression shall be larger than 100 000.

Arithmetical expressions may have the operators +, -, *, /, // and **, the last three representing division, integer division and exponentiation. The operands are partial expressions, variables or constants.

3.4.3 Consequents

(1) Only actions are handled by SMARN

We have parameterized consequents

(O,H,P),

where O is a deontic operator, H an action and P the subjects of the action. Neither deontic operators nor subjects will be handled explicitly in SMARN1. But the subject a given problem is solved for will be represented.

It will be interesting, in later versions, to return to these restrictions and possibly introduce such primitives.

A consequent will be expressed

```
K ::= <statement>/              (cf. SIMULA common base)
      <apply statement>/
      <normblock call>/
      <stochastic consequent>
```

(2) Implementation

The consequents we shall model may contain

- the introduction of operationally new facts. This is achieved by declaring new fact-variables, x, y, z in norm blocks.

- change of fact-values for fact-variables, e.g. legal positions. (This obviously only applies to conventional facts, not to brute facts). This point may be viewed as a special case of the next. Changes in values will be expressed by statements. x:=123+y, shall mean that the variable x is assigned the value of the expression on the right side of the assignment sign, :=, the value being 123 plus the value of y.

- a reference to the further problems that further have to be solved if the consequent is to be realised. This is a reference to a norm-segment that represents the solutions of the actual problems. The references are made by means of a sentence, such as apply S, where S is the name of a norm segment.

(3) Actions and permissions

Actions may be very different from one another. This is true, for instance, with respect to the time aspect of actions. Some actions have immediate consequences when a decision is made, for example in the case of decisions or changes occurring in legal relations or positions. For example an entitlement might be granted immediately according to the decision. Such decisions may be handled by SMARN.

Other decisions prescribe an action which is to take place in the future, at a more specific point of time, for instance, or on more specific conditions. This kind of application needs a real-time-mechanism if it is to be implemented. What SMARN is not intended to include.

Consequences follow when the conditions of a rule are satisfied.

We will differentiate between actions on two levels: for decision-makers and for the clients. To say that something is permitted for the computer requires a mechanism for making choices regarding when a permission is to be used and when not.

In SMARN1 a permission for a client is described as a choice of action. This means that if the client has an opportunity to choose between alternatives, probabilities will be attached to each alternative, reflecting the client´s

choices. This should be done on the basis of expectations or statistics.

Since SMARN1 is a simulation model, this approach is satisfactory. The same is not true for the case of decision-automata. There, a permission for a client must be respected by forcing further decisions building on the clients choice to wait till his choice is reported. SMARN2, therefore, has a mechanism which express merely that something is permitted for a client.

3.4.4 Rule declaration

Rules may be declared in SMARN.

The syntax of a rule declaration is

<rule declaration>::=
rule <name> = R

Conditions may contain calls for weighing, or calculations of internal facts (by type norm-blocks).

3.4.5 Rule call

<identifier> for a rule in a statement list shall be interpreted as a call. Rules may be called in a block and from else-branches in if-statements.

3.5 Norm-segment and norm-blocks

3.5.1 Norm-segment

We shall introduce the concept of a norm-segment meaning thereby a textual contiguous section of norm-descriptions. This text section corresponds to the description of norms for solving a problem P. If P = (x,T,L), the norm-segment will consist of a prepared version of the legal sources, L, where L is a sequence of problems and a list of unstructured facts T.

We also assume that norms are applied sequentially within the segment, if norms do not explicitly state the control of flow, by giving another norm to be applied next.

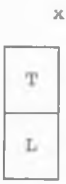
The norm-segment concept is central in SMARN1 and 2. It is based on the hypothesis that norm-descriptions are often arranged in order of application in mass administration cases. But, strictly, the concept presupposes only that norms may be arranged in the sequence in which they will be applied. It does not presuppose that this is done in acts and regulations, etc. Consequently, the hypothesis is that such sequences exist, at least in the sense of time-sequence at the time of application.

From 2.2.3 we know that partial solutions may be named down to the norm level. Solution-descriptions may be explicit or occur as references to explicit solution-descriptions. We shall refer to sequences of norms by name. A name of a norm-sequence, that is the norm-segment, may therefore be used for reference. When solving a problem, in a segment, it may be specified that another segment is to be activated for the solution of a partial problem.

3.5.2 Norm-block

A norm-block is an implementation of a norm-segment. It may be named and contains declarations of fact-variables, corresponding to the introduction of new concepts in the norm-segment. Further, it may contain declarations of weighings and a sequence of rules.

A named block may be represented



$P = (x,T,L)$

x is the name of the block. T are the declarations of the fact-variables and L is a sequence of rules. The order in T is

arbitrary. But in L it is related to the sequence in which the norm-models are executed.

We operate in SMARN1 with three kinds of blocks. One type to solve problems that concern the choice of weighing function (weight norm-block), one type that may assign values to fact-variables (fact norm-block), and a type-free block type.

3.5.3 Norm-block declarations

All facts registered in one norm-block are collected in one schema.

We have three types of norm-blocks: Type-free, type or weight norm-blocks. A type norm-block (function norm-block) assigns a value to a result-variable for the block (Syntax, see SI 10.1.1). A type norm-block is used to evaluate the values of an internal fact, which is a result of a legal decision process, while a weight norm-block decides which of a set of weight-functions is to be applied when weighting a factor.

The type-free norm-block is used to solve general legal problems.

In the definition of norm-blocks reference is made to the definition of <unlabelled block> from the Algol report 4.1.1 (Naur: p. 23).

(1) Named, type-free norm-block

A named type-free norm-block has syntax

```
<named typefree normblock>::=
    normblock <identifier>;
    <unlabelled block>
```

Declarations in the block (SI 10.1.2) may contain rules and weighing declarations, and declarations of fact-variables, as well as SIMULA-declarations of variables, procedures, classes, etc. Body in the norm-block (SI 10.1.5 and SI9) contains a statement list.

(2) Nameless, type-free norm-block

A nameless, type-free norm-block has the same syntax and semantics as a named, type-free norm-block. Except, obviously, for the heading

    normblock <identifier>;

The syntax is then

    <typefree normblock>::=<unlabelled block>

(3) Type norm-block

A type norm-block calculates the value of an internal fact (SI 10.3). The types are boolean, integer, real, text, character and define the result-variable´s type.

    <type normblock>::=
        <type> normblock <identifier>;
        <unlabelled block>

is the syntax of the type norm-block. In the body of the block there should be an assignment statement.

    <variable>:=<expression>;

where the variable is the block name.

(4) Weight norm-block

A weight norm-block defines which weight-function is to be applied to determine a specific factor´s influence in a weighing. The syntax is

```
<weight normblock>::=
    weight normblock <identifier>;
    <unlabelled block>
```

In the norm-block there must be an apply-statement.

3.5.4 Norm-block calls

Here, we shall discuss the calls according to the norm-block's type.

(1) Type-free norm-blocks

Type-free, unnamed norm-blocks are very similar to SIMULA-blocks, and cannot be called. If a type-free block is named, it may be called from a statement position. An unnamed block may also be placed in a statement list, where SMARN-statements may be located.

(2) Type norm-blocks

Type calls may be located in type-expressions and in weighings by def <normblock identifier>.

(3) Weight norm-block

Weight norm-blocks are called only when evaluating weighings. For any factor this is done by the block specified

    func <normblock identifier>.

3.6 Weighing

The guidelines describing a weighing are to be grouped together in a set of norms.

We shall make no assumptions about the sequence in which the guidelines are applied within the same set of norms for a

weighing. We assume only that from the moment a weighing is started, and until it is finished, only guidelines attached to the specific weighing are used.

The construction of a reasonably good weighing model, even for only simple cases, is one of the greatest challenges of the project. We have acquired some knowledge of the problem through the SARA project (Hansen: 1981a) and the preparatory project for SMARN (Hansen:1981b).

The original weighing-model was as follows: Weighings were described partly by a set of factors. Some pull in the positive direction, some in the negative. It is assumed that one may find factors which are such that the weight of each factor is relatively constant from one case to another. It is also assumed that any factor either occurs, does not occur or is irrelevant in any particular case. At the outset, this model - called the weight-number rule (Eckhoff:1975, p. 150) - was planned to be used as a weighing model.

Both in the SARA report and in SMARN's preliminary report, there is argued that a more general model should be used to model weighings. The main reason is that the assumption about finding factors with constant weights is dubious. It is possible to use the model, but with a largely reduced efficiency. For, if a proposed factor implies strongly varying weights from case to case, several factors have to be introduced to substitute it in order to retain the assumption of constant weights.

This criticism led to a more general model. Exercising a discretionary decision was here described by means of a weighing and a process measuring out the consequents. The weighing is described by

- a fixed set of possibly relevant factors, each of which is characterized by,
  - a quantifiable measure,
  - the weight, Wx1, expressed as a function of this measure
- the total weight for the factors is described as the

sum of the weights for each factor.

To be able to describe certain kinds of dependencies between factors, alternative weight-functions were introduced for the same factor. Depending on which situation occurred in a particular case, different weight-functions could be used. Likewise, a fact (quantifiable measure) could be the result of an earlier decision, PCyj. To decide which function to use, a decision PWzk could be executed.

For a specific weighing-problem, the weighing yields a weight for the case. The result is measured on the basis of this weight and other facts.

Temporary results from the weighing process must not be used.

The measurment of the result will be carried out via the application of a rule.

The complete model of a weighing will then consist of five components,

Px = (x, Fx , Wx , PCy , PWz).

PCy is a set of problems connected to the determination of fact-values. PCyj corresponds to Fxj. PWz is a set of problems connected to the decision concerning which weight-function is to be used. PWzj is attached to Wxj, etc. Wx, PCy og PWz correspond to Rx in the problem-model.

3.6.1 Declaration of weighing

The syntax for a weighing is

<weighing declaration>::=
    weighing <identifier> = <mlist>

<mlist>::= <mlist>,<m> /
           <m>

<m>::=<identifier> /

(<identifier>, def <identifier>)/
(<identifier>, func <identifier>)/
(<identifier>, def <identifier>, func <identifier>)

<identifier> alone refers to a fact-variable, def <identifier> refers to a type norm-block for assigning the value to the variable and func <identifier> refers to a weight norm-block to decide which weight-function to use.

<m>'s identifier must be specified punk.

Example:

Weighing x=a,b,(c,def d, func f);

This weighing contains three factors a,b and c. c's value can be assigned by d if the user so wishes. f decides which weight-function is to be used for the factor c.

3.6.2 Weighing call

<identifier> for a weighing is to be interpreted as a call. A weighing call may be located where the type-block calls are located.

3.7 Control-structure

Some control-structures will be necessary. Basically, these are techniques for skipping a sequence of norms, that is, for bypassing it without activating it. Alternatively, this problem could be solved by manipulating the antecedents of the norms that are to be ignored. But this may be a tedious solution. The problem sounds perhaps a little artificial, and it is worth noting that it is forced on us by the norm-segment concept.

3.8 Schemata

Associated facts form a schemata. As a rule one schemata will be associated with each norm-segment, and therefore also to each norm-block. The schemata concept will be a central concept when SMARN1 is used to register facts.

3.9 Using SMARN1

The use of SMARN will involve programming, initiation and case handling.

3.9.1 Programming

The process involving the transformation of a set of rules and guidelines into SMARN formalism will be called "programing in SMARN". It is a task for the users of the language, not for the developers.

3.9.2 Preprocessing

SMARN programs will be translated into SIMULA before execution.

Originally, the design and implementation of the preprocessor was held to be a central task for the project. While the design is relatively detailed, the task of implementation was stopped due to the extension of SMARN1 to SMARN2.

Preprocessing is an automatic process.

3.9.3 Initiation

A set of norms described in SMARN must be initiated with case-independent or norm-specific facts before a SMARN program may be run. This is a task for an end-user. Initiation will be controlled by the SMARN program.

3.9.4 End-user's commands

We will now sketch a command language available to the end-user of SMARN. In addition to the outlined commands, one would prefer to allow the end-user to manipulate the facts arithmetically, turning SMARN in the direction of a 4th-generation language. This is probably a correct strategy, but has not been included in the current specifications, because of increased implementation complexity.

In 3.9.4 <problem> may be a superior problem or a sub-problem. In the first case it is identified by the norm-block name, in the other by a unique path down to the sub-problem. Examples may be

P and
P.Q.R,

where P is superior and R is a sub-problem of Q, itself a sub-problem of P. If there is only one sub-problem R of P, one may refer to R by

P.R.

The syntax for <problem> will be

<problem> ::= <problem name> /
<problem name>.<problem>

<cases> will be a set of cases for a given problem-type.
A case is referred to by a unique key, <key>.
A set of cases of the same type may be given a name by the NAME command. These names must be unique.
Cases may also be specified in the form

<problem> WHERE <Boolean expression>

where <Boolean expression> has the syntax

<Boolean expression> ::= <attribute expression> /
not <Boolean expression> /
<attribute expression> and <Boolean expression> /
<attribute expression> or <Boolean expression>

and where <attribute expression> has the syntax

<attribute expression> ::=
<attribute> <comparator> <arithmetic expression>

<comparator> is the six standard comparators, while <arithmetic expression> is stated on variables in a norm-block. The conditions may only be stated within the same norm-block.

<cases> will have the syntax

<cases> ::= <case> / <name of cases> /
<problem> WHERE <Boolean expression>

<case> ::= <problem>,<key>

(1) INIT

INIT <problem>,<key>

starts initiation of a case with key <key> of the type <problem>.

All subordinate decisions under the given one that must be solved to solve the case must also be initiated.

An incomplete registration is not accepted. This means that one must register the facts for subordinate decisions in the sequence they are presented. For all solutions other than this one has to rely upon coincidence to get the proper set of facts for an actual decision. The reason is that the values of the variables will be uncertain at every point after the point where one ended registration, with the consequence that redundant information might be registered or relevant information lacking.

One may, however, interrupt the registration process, for a later restart.

(2) CHANGE

CHANGE <cases>

may be used to change the values of facts in <cases>. Values are changed by entering a

c - for change together with the variable name, for as long as one wishes to change values in the case,
e - for end.

There now arises a possibility that further sub-cases must be initiated. Since a value of a variable has been changed, it might be that some of the old sub-cases are redundant, whilst some new ones might also be needed.

(3) SOLVE

When the initiation of the program is finished, the computations may start. A central command will be

SOLVE <cases>, or
SOLVE <problem>,<poplist>.

<poplist> is a list of cases associated with one problem, specifying the number of occurences of each case-type. The syntax is

<poplist> ::= (<case>,<POP-switch>) /
(<case>,<POP-switch>),<poplist>)

<POP-switch> is defined in (10).

For each partial problem, the schemata - with case specific facts - must be entered (or already be given), before the partial problem can be solved.

(4) MACRO

A macro is a list of commands given a name. One may call upon the execution of the command list by the DO command and the macro's name.

The following is a macro solving problems P and Q for populations of the size 1000 and 200, respectively.

```
MACRO <name>
SOLVE P, POP = 1000
DISPLAY('THIS IS THE TOTAL EXPENCES FROM P';)
DISPLAY(X;)
SOLVE Q, POP = 200
DISPLAY('THIS IS THE TOTAL EXPENCES FROM Q';)
DISPLAY(Y;)
DISPLAY('GRAND TOTAL';)
END <name>
```

(5) END

END <name> ends a macro with name <name>.

(6) SAVE

SAVE <name> saves a macro <name> on external storage. The macro will then be available for later use.

(7) DO

DO <name> starts the macro named <name>.

(8) LIST

```
LIST PROBLEM
```

lists names of described decisions on the most abstract level.

```
LIST POP or LIST POP <name>
```

lists names of population descriptions or a given population.

LIST MACRO or LIST MACRO <name>

lists names of macros or a given macro.

(9) GET

GET <cases>

retrieves <case> of type <problem> for the purpose of reading or printout.

(10) DISPLAY

Accumulated specified variables may be displayed:

DISPLAY(X) - displays the variable X
DISPLAY(X,Y) - displays X and Y on the same line
DISPLAY(X, ,Y) - displays X and Y separated by four blanks on the same line
DISPLAY('XY') - displays the character sequence XY
DISPLAY(X;Y;Z;) - displays X, Y and Z on three lines

(11) The switch POP to the SAVE command

If statistical calculations are selected, one will need to give the sizes of the actual populations. This is done on the command level by a switch. The syntax for this switch is

<POP-switch> ::= POP = <constant>

The population for a problem P s case-type C is set to 1000 as follows:

SOLVE P,C,POP=1000

(12) NAME

It will be possible to assign a name to a set of cases. This is done in one of the following ways:

```
CASE NAME <name> = CURRENT <problem>
                 = ALL <problem>
                 = <name1> + <name2>
                 = <name1> - <name2>
                 = <name1> * <name2>
```

The name is assigned to current cases for a problem, all cases of a given problem, and the union, difference and intersection of two sets of cases, respectively.

It is also possible to assign a name to a list of populations associated with a given problem,

```
POP NAME <name> = <poplist>
```

(13) REM

One can remove certain cases by using the REM command. All sub-cases of a case that is removed will also be removed.

3.9.5 Fact base

A very simple fact-storage is described, which is such that the facts may be stored, retrieved, used and changed on later occasions.

4 AMENDMENTS FROM SMARN1 TO SMARN2

4.1 Introducing objects and sets

4.1.1 The "client-department" model in SMARN1 without object and relation concepts

Originally, we restricted our modelling efforts to situations in which a client was brought into contact with a public service or department. Hence the name "client-department" model. Within this field we aimed at simulating effects of a set of norms. Furthermore, the problems to be described in SMARN1 have to be in a schematic form, with respect both to the description of facts and the norm-expressions regulating the solution. But then the further details of the problem might be relatively arbitrary. For instance, the subject may be the client's economic entitlements from the department, other entitlements, duties on the client's part, etc.

The fact-model for this domain is a rather simple one, as sketched above. It is not unnatural to think of the client as one object and the department as another. For a given problem, facts describing the client, or associated with him, and facts associated with the department, form two groups. One kind of facts is called case-specific, the other norm-specific.

The facts are not structured in any other way than by conditional structures, an example being that only if a person is married is it possible to give information about the spouse. With the exception of conditional structures, the fact-structure for a given problem is flat. It could be viewed as a list of facts.

Additionally, we have seen no reason for introducing a relation concept in the "client-department" model in SMARN1. There is one important relation in this, the one characterizing the model itself, namely the relation between

the client and the department. We chose to regard other relations on the client or the department side as facts attached to client or department.

4.1.2 The Inheritance Act is not describable in the "client- department" model

The Inheritance Act was chosen in the autumn of 1982 as our first problem domain for description in SMARN. It was immediately clear that it became necessary to introduce both an object and a relation concept, the latter in a slightly transformed form as a set concept.

In inheritance cases it is often necessary to describe a group of persons entitled to a part of the inheritance. Each one of these entitled persons must be described by certain facts. It is convenient to describe these persons in separate, similar data elements, called objects.

Likewise, the relationships between the persons are of interest. These two extensions led to a comprehensive change of SMARN1´s basic concepts.

4.1.3 Object

For our purpose, we may think of an object-model as a collection of facts associated with a physical object. We will normally refer to the object-model as an object. An object may contain variables (atomic facts) and references (which are strictly speaking variables as well) to associated objects, elements and sets.

4.1.4 Relation

Usually, relations are understood as relations beween two objects. Mathematically, however, one can define relations between an arbitrary number of object-types, from one upwards. We shall define a concept of this kind. If x and y are person-objects, and z is a property object, typical relations could be

x fatherof y,
x owns z and
soccerteam(x1,x2, .., x11).

4.1.5 Set and element

The reason for introducing objects was the need for simultaneous representation of equal, physical objects in one and the same case. In this situation, one may speak about a set of objects.

It will often be desirable to describe elements in sets more efficiently than by means of objects. For example, we may be interested in establishing the set of married persons. And, as we know, such a set will contain elements with two objects in each.

Each element may generally contain an arbitrary number of objects, also of the same type. But then the objects should be separable by their role in the element, e.g. man and woman in the example of the set of married persons.

4.1.6 A relation may be expressed by a set

If M is a set of males and K is a set of females, then "m married to k", m ε M, k ε K, (ε: belongs to) may be true for one and only one couple (m,k), given m or k, a monogamous society and legal states. For (m,k), the relation "married to" is then satisfied.

All relations may be described by sets. For example, instead of describing a married-to-relation one may operate with a set of married couples. Instead of saying that m is married to k, we shall say that m and k are married. That is, (m,k) ε G, where G is the set of married couples.

4.1.7 The choice of sets for expressing relations

Operating with both a set and a relation concept, one will need operators for both representations and for transforming sets to relations and vice versa. One economizes with

constructs by introducing only one or the other type of concept. But the price is that one is forced to think of relations as sets: a's being married to b becomes the married couple a and b. This is a severe limitation, if not in expressional force, then on a mental or psychological level.

4.2 The object and set concepts extend the fact-model

4.2.1 The original fact-model

This model has only one kind of element, namely variables. These can be binary, integer, real or text. For a given case the variables have their definite values assigned. The facts are associated with problems. Thereby, the problem structure defines the fact structure.

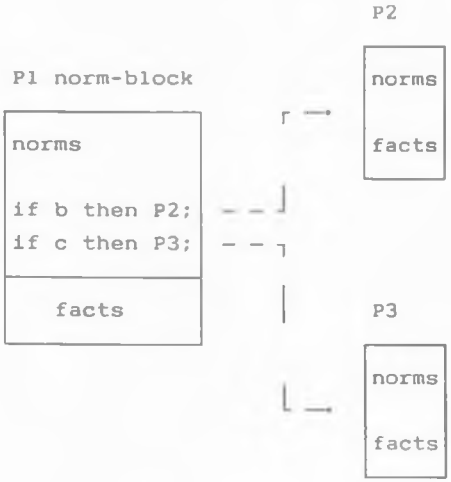


Figure 4-1: Fact and norm-block structure in SMARN1.

P1 is the main problem, while P2 and P3 are sub-problems.

The only additional structure to the problem structure as imposed on the facts, is the conditional fact structure.

4.2.2 SMARN2´s extended fact-model

The object and set concepts increase the expressive capability of SMARN. Objects may contain atomic facts, references to associated objects, elements or sets. Domains define the element-types and thereby also the set-types. If several equal object-types are used in an element, they must be separated from each other and given names according to their role in the element, cf. the example above with "man" and "woman" for elements of the set of married persons.

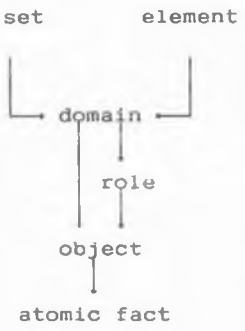


Figure 4-2: Connections between basic concepts in the fact-model

a ⟶ b is here read ´a is defined by b´.

4.2.3 Example of a fact-model

For some reason or another, we may be interested in the name and age of a person, what cars he owns and who his mother, father and children are. If we form an object called "person", with atomic facts "age" and "name" and with reference to persons called "mother", "father", to a set of persons called "children" and to a set of the "cars" the person owns, then -on a sufficiently vague level - we have solved the problem of

representing the facts.

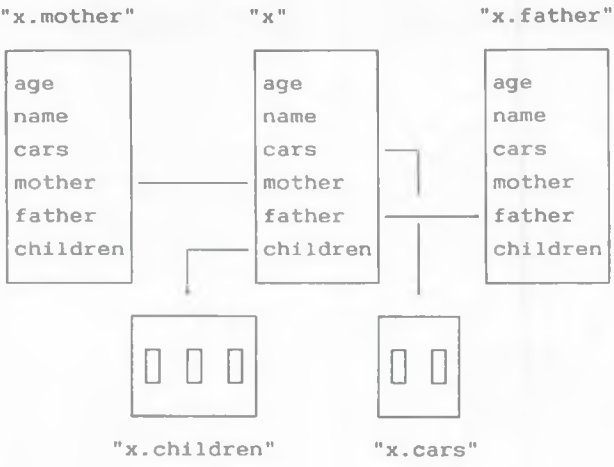


Figure 4-3: Person-objects with relationships and attributes

4.3 Introducing object and relation concepts lead to the description of the solution of a problem by a procedure

4.3.1 Repetitive object-type

The fact that the same problem may occur several times represents a situation that could not arise in the "client-department" model. A problem occurence could there be written

$$Px = (x, Fx, Cx, Rx),$$

where Cx are the variables associated with the fact-types Fx. If Fx is described in the extended model by an object-type, it is obvious that we may have several occurrences of the variables Cx for Fx, Cx1, Cx2, ..., Cxm, where m is the number of occurences of the problem Px. We therefore need to identify the specific occurrence y of a problem-type we wish to refer

the specific occurrence y of a problem-type we wish to refer to in a particular case

$$Pxy = (x, Fx, Cxy, Rx).$$

4.3.2 The hierarchy restriction for the solution of problems is too strong when equal problems occur several times

The hierarchy restriction was not especially severe in the "client-department" model. Since the problem P could occur only once in the same case, a problem definition

$$P = (\ldots , P, \ldots)$$

would at best represent the fact that identical problems were solved several times. But solving identical problems several times does not seem wise, since one may use the first solution in the next ones. However, a recursion could simulate refinement of a solution. So the hierarchy restriction created a certain limitation in the "client-department" model, too.

In the extended model, it is not desirable to accept the hierarchy restriction. It would here represent a more severe additional restriction than in the "client-department" model. A problem-type may very well be defined here by itself, i.e., recursively. A particular instance of a problem will not, however, be defined by itself.

An example of a problem-type of this kind concerns how one is to determine the group of persons who are to inherit the estate of a deceased person. If the person had living successors, the inheritance would belong to the children. Or, if some of the children were already dead on the day of the death of the parent, these childrens' children would inherit. Establishing the group of beneficiaries to a persons will often involves resolving the same issue for some of this person's children, etc. So here is a recursive problem.

4.3.3 Procedures take the norm-blocks' role as the description of the solution of a problem

Norm-blocks were introduced in a context in which the hierarchy restriction was not too rigid. When the hierarchy restriction is loosened, another problem-model must be established. Since, in principle, we are interested in handling recursive problems, a procedure is a suitable good candidate in ALGOL-like languages. This is the reason for choosing such an implementation. But it may very well be interesting to extend the procedure concept of SIMULA, for example by allowing a variable number of parameters.

In the procedures, only temporary variables and references to objects, elements and sets are permitted.

4.4 A schemata is associated with an object

While the schemata concept in the "client-department" model in SMARN1 was attached to norm-blocks, this is not possible in SMARN2, simply because that concept is removed. It will obviously not be desirable to connect schemata to procedures either. The chosen solution is to connect it to objects.

4.5 Generalised weighing-model

Even the generalised model has its weaknesses, although it does generalise the weight-number rule. The weights of factors are described as a function of one quantifiable measure (function of one variable).

A weakness of model 2 is that the number of factors is fixed for each weighing model. The relevance-evaluation of a factor may then be expressed in one of two ways, either as a choice between alternatives, or by introducing a special irrelevance code for the quantifiable measure in the weighing functions. Neither of these solutions is at all elegant.

Another weakness is that the weighing markers in the antecedent for guidelines about relevance are very difficult

to handle.

A third weakness is that while rules and guidelines have the same form, i.e.,

if B then K,

the forms of rules and guidelines in model 2 are very different.

This criticism leads to two generalisations. First, one may have arbitrarily many arguments for each weight-function (functions of several variables), instead of only one as in model 2. Then, guidelines will be represented in the form

if B then K.

Examples:

Let f, f1 and f2 be factors whose weights depend on three facts a, b and c.

-about relevance:

```
if d then w:=w + f(a,b,c),
```

-about weight:

```
if d then w:=w + f1(a,b,c)
else w:=w + f2(a,b,c);
```

-about value:

```
if d and positivevalue then w:=w + f(a,b,c)
else if d and negativevalue then w:=w - f(a,b,c);
```

The whole weighing may alternatively be thought of as a set of variables, or - as is the case in the example above - as the sum of the weights produced by the factors' weight-functions (also of several variables).

A further generalisation is also possible. That consists of expressing the weights in terms of relations instead of functions. (A function is recognized by the fact that a given set of values for the function's variables, results in one and only one value for the function. While there may be several values for a relation, given a set of values for the variables.) The reasons for choosing functions instead of relations are of a practical nature only: it is more efficient to handle functions in the long run.

5 SMARN2 DESCRIPTION

In SMARN2, procedures will express solutions of problems while facts will be described by SMARN-objects and sets. The procedure will be used in four different ways:

- solving general legal problems - typeless,
- solving discretionary decisions - type procedure,
- describing a factor in a discretionary decision - real procedure,
- technical calculations may be described together with the legal ones. They do not need to be put in separate procedures, but may obviously be separated if that is convenient.

In addition to the procedures, a marker-variable will be central. It associates a variable with a weighing. The variable may either have its value set manually, or by executing the discretionary decision.

5.1 SIMULA extension

5.1.1 Declaration

Syntax

```
<declaration>::=<SIMULA declaration>/
                <marker declaration>/
                <role declaration>/
                <domain declaration>
```

Semantics

<declaration> (Dahl:16, Naur:23) is extended by marker, role and domain declarations.

This obviously means that marker and domain may be declared whereever a declaration may be placed in SIMULA.

5.1.2 Type

Syntax

```
<type>::=<SIMULA value type>/
         <reference type>/
         <set type>
```

Semantics

<type> (Dahl:27) is extended by <set type>. Note the consequences the <type>-extension has, e.g. through the <type> procedure in SIMULA (Dahl:57). Since <type> is extended by pointers oref to SMARN objects and eref to elements as well as set-type to handle set,

```
oref(o) procedure x; ...;
eref(d) procedure y; ...;
set(d) procedure  z; ...;
```

will be type procedures for object and element references and set. (o is an object type and d is a domain for a set). Accordingly through <type> array (Dahl:57)

```
oref(o) array a ..
eref(d) array b ..
set(d)  array c ..
```

are introduced.

Examples

A set procedure to establish the union of two sets will look like:

```
set(d) procedure s(a,b);
set(d) a,b;
s:=a+b;
```

An eref procedure to establish a reference to the last element in a set may be written

```
eref(d) procedure y(s);
set(d)s;
y:-s.last;
```

An array, z, of references to objects of type o with length 14 will be declared

```
oref(o)array z(1:14);
```

5.1.3 Reference type

Syntax

```
<reference type>::=<SIMULA reference type>/
                   <SMARN reference type>
```

Semantics

<reference type> (Dahl:27) is extended by <SMARN reference type>.

5.1.4 Assignment statement

Syntax

```
<assignment statement>::=<SIMULA assignment statement>/
                         <set assignment statement>
```

Semantics

Assignment statement (Dahl:42) is also extended.
Since SIMULA's type is extended, the assignment statement

will be extended accordingly.

Examples

```
s:=a;
s:=a+b-c;
s:=(e1,e2,e3);
s:=;
```

are all legal set expressions that may be assigned to set variable s. A set variable that is to be assigned to a set expression must have a domain identical to that of the set expression.

5.1.5 Reference expression

Syntax

```
<reference expression>::=<SIMULA reference expression>/
                         <SMARN reference expression>

<SMARN reference expression> ::=<SMARN object
                                 expression>/
                                 <element expression>
```

Semantics

<reference expression> (Dahl:34) has been extended by SMARN object expression and element expression.

Note the consequences with respect to SIMULA's reference assignment (Dahl:42) and for clause (Dahl:46).

Example

```
for e:-e1,e2 do e.person.income:=0;
```

Assuming e, e1 and e2's domain contain an object "person" with an attribute "income", this for statement has the effect of zero-ing the income of e1 and e2's persons.

5.1.6 For clause

Syntax

<for clause>::=<SIMULA for clause>/
for <set expression> do/
for <set expression> while <Boolean expression> do

Semantics

The point of this construct is to evaluate the <controlled statement> (Dahl:46) for all elements in the set. If set expression is not a set variable, then a temporary set variable, call it T, is established behind the curtains, T:=<set expression>.

The statement may then be written

for T do <statement>.

To save explicit inspections down the object, element and set path, implicit inspections will be supplied in the order of set occurrences, element occurrences, tuple occurrences, etc. An implicit inspection will be supplied only for sub-structures of the set expression which is being processed. It will be defined by the missing path down to a referred entity, and with the <controlled statement> as scope.

Example

Let us assume that two sets, s and c will be used to produce a set b. The process is to terminate when all elements in c are contained in s U b, or when there are no more elements of s left to process. This may be stated:

```
for s while (s+b)*c<c do ...;
```

5.1.7 Statements

Syntax

```
<statement>::=<SIMULA statement>/
              <weighing statement>/
              <set handling>/
              <object handling>
```

Semantics

In addition to those in SIMULA, <statement> (Dahl:42) may be <weighing statement>, <set handling> and <object handling>. The weighing statement is a call for a weighing. This will not be executed if the value of the weighing is set manually. <set handling> and <object handling> contain mechanisms to store on disc (put), load from disc (get), initiate (init) and totally remove (rem) objects and sets, respectively.

5.1.8 Relation

Syntax

```
<relation>::=<SIMULA relation>/ <set relation>

<reference relation>::=<SIMULA reference relation>/
              <element reference relation>/
              <SMARN object reference relation>

<element reference relation>::=<simple element expression>
            <reference comparator>
            <simple element expression>

<SMARN object reference relation>::=
            <simple SMARN object expression>
            <reference comparator>
            <simple SMARN object expression>
```

```
<set relation> ::= <simple set expression>
                   <reference comparator>
                   <simple set expression>
```

Semantics

The meanings of the relational operators for sets are as follows:

```
B < A,      B is included in A,
B ≤ A,      B is included in or equal to A,
B = A,      B is equal to A,
B > A,      A is included in B
B ≥ A,      A is included in or equal to B
B <> A,     A is not equal to B
```

Two element (object) references A and B are identical if they refer to the same element (object) or if both are none (refers nothing). In this case "A==B" is true.

```
A=/= B = not (A==B)
```

5.1.9 Remote identifier

Syntax

```
<remote identifier>::=<SIMULA remote identifier>/
                       <simple element expression>.
                       <role identifier>/
                       <simple element expression>.
                       <attribute identifier>/
                       <simple SMARN object expression>.
                       <attribute identifier>
```

Semantics

Let E be a simple element expression qualified by a domain D and let O be an object identifier and R a role identifier in

the domain. Then

E.O and E.R

are remote object and role identifiers, respectively. If S is a simple SMARN object expression, and A an appropriate attribute then

S.A

is a remote identifier. If the expression part evaluates to none, a run-time error will occur.

Examples

If e is an element reference variable qualified by

domain(man,woman) couple; then
e.man

is a valid remote identifier. If op is a reference variable qualified by SMARN object o, oref(o)op, and if o has an attribute property, then

op.property

is a valid remote identifier.

5.2 SMARN program

Syntax

<SMARN program>::=<SMARN block>/
<compound statement>

<SMARN block>::=<SIMULA block>/
<SMARN unlabelled block>/
<label>:<SMARN block>

```
<SMARN unlabelled block>::=<SMARN block head>;
                            <compound tail>

<SMARN block head>::=begin <global declaration>/
            <SMARN block head>;<global declaration>

<global declartion>::=<declaration>/
                      <object declaration>
```

Semantics

A SMARN program will consist of a list of global declarations between begin and end. These are of two kinds,

- <declaration> and
- <object declaration>

The procedures are caught by <declaration>, the facts by <object declaration>, while object, element and set pointers are caught by the <SMARN type> and <set type> declarations.

5.3 SMARN's type declarations

Syntax

```
<SMARN value type>::=result<SMARN long type>/
                     dynamic <SMARN long type>/
                     temporary <SMARN short type>/
                     <SMARN long type>

<SMARN long type>::=real(<ndigits>,<ndecimals>)/
                    integer(<ndigits>)/
                    text(<ncharacters>)/
                    boolean

<SMARN type>::=<SMARN value type>/
               <set type>/
               <SMARN reference type>
```

<SMARN reference type>::=<element type>/
<SMARN object reference type>

<SMARN short type>::=integer/real/boolean/text

Semantics

<SMARN type> has three sub-types: <SMARN value type>, <set type> and <reference type>. The value type may only be used in SMARN objects, while set and reference types are contained in the extended SIMULA type definition. I.e., sets, references to objects and elements may be declared both globally, in procedures and objects.

<SMARN long type> may have the specification "result" or "dynamic" to indicate that the variable is an output or conditional variable, respectively. A <SMARN short type> variable must be specified "temporary" to indicate that it is a temporary variable.

ndigits, ndecimals and ncharacters are numbers of digits, decimals and characters.

<SMARN value type> variables may only be declared within objects.

SMARN value type variables may be specified differently according to their different roles. Two points of time are important to the differentiation: the point of registration of the facts and the point of storing the facts.

With respect to registration, default is that the values of all the variables in the objects are asked for. If this is not desirable, that may be for three kinds of reasons: the fact is not of input but of output (result) kind, or the fact is interesting only if specific conditions among the facts occur (dynamic kind), or the variable may be temporary. In the last case it is only used as scratch.

With respect to storing, default is that the values are stored in the object. However, this will not take place for variables specified temporary.

Effect of SMARN value type variable-specification:

specification type	registered at registration time	stored at storing time
no specification	yes	yes
temporary	no	no
dynamic	conditional	yes
result	no	yes

Example

A real, p, of length 7 characters with 3 decimals is declared

real p(7,3).

5.4 Marker declaration and weighing statement

Syntax

<marker declaration>::= <SMARN short type>marker
(<procedure identifier>)<identifier>

<weighing statement>::=<marker identifier>(<actual parameter part>)

Semantics

A marker (abbreviation for weighing marker) is a variable associated with a procedure. Calling the weighing will have the effect of

- either manual assignment of the marker's value, or
- evaluating the procedure and assigning the value of the procedure to the variable.

This construction may be regarded as peculiar, but is needed to give the preprocessor the opportunity to make a distinction between ordinary type procedures and weighing procedures.

Example

If the procedure is declared

```
integer procedure a(p); ..;
```

then the marker may be declared

```
integer marker (a)m;
```

If q´s type is compatible with p, then m(q) is an example of a weighing statement.

5.5 Object

5.5.1 Object declaration

Syntax

```
<object declaration>::=<object specification>
                       <object head><object tail>

<object head>::=object <identifier>;
                       begin <objectlocal type declarations>

<object tail>::=end/<dynamic initiation>;<object tail>/
                       <outfile call>;<object tail>

<dynamic initiation>::=<if clause><object initiate>/
                        <if clause><set initiate>/
                        <object initiate>/
                        <set initiate>
```

Semantics

<identifier> is the name of the defined class of objects. <objectlocal type declarations>, described in 5.5.5, is a list of declarations of sets, element variables, object reference variables, etc.

The <dynamic initiation> and the <outfile call> are the only legal statements in the object declaration.

<outfile call> refers to a call to a procedure in SIMULA's outfile class. This may be used to produce leading text to guide the registration process.

Example

```
object c;
begin
    oref(c)x;
    set(e)s;
    domain(d)e;
    real(5,2)z;
end;
```

Object c contains an object reference x to c-objects, a reference to a set of elements from domain e, called s, and finally a real z with 5 digits, two of which are decimals.

Conditional object initiation is permitted, e.g.

```
if b then o.init(p)
```

as well as the unconditional

```
o.init(p).
```

5.5.2 Object expression

Syntax

<SMARN object expression>::=<simple SMARN object expression>/
<if clause><simple SMARN object expression>
else <SMARN object expression>

<simple SMARN object expression>::=none/<variable>/
<function designator>/
<simple element expression>.
<SMARN object identifier>/
<simple element expression>.
<role identifier>

Semantics

A simple SMARN object expression is of type

oref(<qualification>).

The value of the expression is a reference to an object or it is none. In a conditional SMARN object expression, both alternatives must have an identical qualification.

Examples

Let a and o be objects and let

```
oref(o)op;
oref(o)procedure fp;
eref(d)see;
domain(o,r)d;
role(a)r;
```

be declared. Then

```
none
op,
fp,
see.o,
see.r,
if b then op else fp
```

are all SMARN object expressions.

5.5.3 Object reference declaration

Syntax

```
<SMARN object reference type>::=link oref
                (<object qualification>)/
                oref(<object qualification>)
```

Semantics

If the object reference is to be used to initiate (init) an object, it must be specified "link". A link referenced object may also be stored on external memory (put), loaded from external memory (get) and removed (rem). If the object reference is not link specified, it cannot be used to init, put, get or rem.

Example

A reference, p, to an object of class q is declared

```
oref(q)p;
```

5.5.4 Object handling

Syntax

```
<object handling>::=<object initiate>/
                    <object get>/
```

<object put>/
<object remove>

<object initiate>::=<object identifier>.init
(<SMARN object reference variable>)

<object get>::=<object identifier>.get
(<SMARN object reference variable>)

<object put>::=<object identifier>.put
(<SMARN object reference variable>)

<object remove>::=<object identifier>.remove
(<SMARN object reference variable>)

Semantics

Initiation, storing, loading and removal of objects may only take place along link-specified references. Objects of the type o, referred to by the pointer oref(o)p, are

- inititated by o.init(p),
- stored on external memory by o.put(p),
- loaded by o.get(p) and
- removed totally by o.rem(p).

To initiate an object means to initiate all of its variables which are not specified result or temporary, and - in the case of a conditional kind - to initiate the variable if the condition is satisfied. The object name (e.g. "person") will be used to produce a heading:

FORM person

This signals the initiation process for the object person in the case at hand.

For each variable (e.g. weight) the name and an equality sign will be output:

weight=

The corresponding value is supposed to be input. Leading text may be used for explanation.

To load an object means to read in values from external storage and to the established internal object. The address on external storage is implicitly given by the internal object reference.

The load-operation on an object only loads one object.

To store an object means to write the object on external storage (external address is implicitly given). All the sub-structure of an object is stored with the object.

To remove an object means to delete all the object´s values on external and internal storage. This also affects the sub-structure of the object.

5.5.5 Objectlocal type declaration

Syntax

<objectlocal type declarations>::=<objectlocal type declarations>;
<objectlocal type declaration>/
<objectlocal type declaration>

<objectlocal type declaration>::=<SMARN type><type list>

Semantics

Within SMARN objects, all SMARN types may be used in full. These are SMARN´s value and reference types, in addition to the set type.

Examples

```
object o;
begin
    result integer (5)p;
    dynamic boolean q;
    temporary real (7,2)r;
    real (7,2)s;
    set (d)t;
    eref (d)e;
    oref (o)op;
    .
    .
end;
```

Between begin and end a number of examples are listed of valid SMARN type declarations local to objects.

5.5.6 Object specification

Syntax

<object specification>::=case/norm

Semantics

Objects may be specified "case" if they have case-specific contents. Otherwise, the contents are assumed to be norm-specific.

Example

A case-specified object named o will have the heading

```
case object o; ...;
```

5.6 Set

5.6.1 Set declaration

Syntax

<set declaration>::=link<set type><type list>/
<set type><type list>

<set type>::=set(<domain identifier>)

Semantics

A set of elements is defined over a certain domain. The domain is a number of object types. A SMARN set will have an ordering which is the order of arrival into the set.

A link-specified set may be initiated, stored, loaded, and removed (cfr. 5.6.8). If it is not link-specified, a set expression may be assigned to it.

Example

A set x of elements from the domain y is declared

set(y) x;

5.6.2 Set manipulator

Syntax

<set manipulator>::=FIRST/LAST/ACT/PREV/PRED/SUC/NEXT

Semantics

Seven operators are defined that can be used on sets: FIRST, LAST, NEXT, SUC(cessor), ACT(ual), PRED(ecessor) and PREV(ious). FIRST and LAST refer to the first and last elements in the set. ACT will refer to the element of the set actually being processed, while the PRED will be the ACT's

predecessor and the SUC its successor. NEXT and PREV will have the effect of changing the actual element to its predecessor or successor, respectively.

Example

For a set s,

```
s.first
```

means s´s first element.

A piece of program to process all elements in a set s could be:

```
s.act:-s.first;
while s.act=/= none do
begin
    "process element";
     s.next;
end;
```

Equivalently, however:

```
for s do "process element";
```

5.6.3 Set expression

Syntax

```
<set expression>::=<simple set expression>/
                   <if clause><simple set expression>
                   else <set expression>

<simple set expression>::=<cartesian product>
```

Semantics

Sets may be combined to form expressions. The operators SMARN

offers are the traditional,

+, to mean union,
-, to mean minus,
*, to mean intersection and
&, to mean cartesian product.

If s and t are two sets defined over domain d, then

if b then s else t

is a set expression.

To be valid, set expressions must be qualified over compatible domains.

Example

set(person) married, nochildren, hascar, hashut:

The set of all persons without any children, who own a car, may be expressed

nochildren*hascar.

The set of all married couples with children, owning a car, but no hut:

(married - nochildren) * hascar - hashut

5.6.4 Set assignment statement

Syntax

<set assignment statement>::=<set left part>:=
<set right part>

<set left part>::= <variable>/<procedure identifier>

<set right part>::= <set expression>/ <set assignment statement>

Semantics

A set expression may be assigned to a set variable qualified by a compatible domain. This variable must not, however, be link-specified.

Example

If sets a, b and s are declared

```
set(c)s,a,b;
```

s can be assigned an arbitrary set of c elements, for example

```
s:=a+b;
```

5.6.5 Set term

<set term>::=<selection>/
<set term>*<selection>/
ø

5.6.6 Set factor

<set factor>::=(<set expression>)/
<set variable>/
<function designator>

5.6.7 Cartesian product

Syntax

<cartesian product>::=<cartesian product>&
<cartesian factor>/
<cartesian factor>

Example

Let

```
object person;
begin
    integer(3)age;
    oref(person)spouse;
end;
domain(person)d;
set(d)men,women,s;
```

be given.

```
men&women where men.spouse==women.person
```

is now the set of married couples.

5.6.8 Set handling

Syntax

```
<set handling>::=<set initiate>/
                 <set get>/<set put>/<set remove>

<set initiate>::=<domain identifier>.init(<set
                  variable>)

<set get>::=<domain identifier>.get(<set
                  variable>)

<set put>::=<domain identifier>.put(<set
                  variable>)

<set remove>::=<domain identifier>.rem(<set
                  variable>)
```

Semantics

A set s declared link set(d) s may be initiated, loaded, stored and removed by the statements

d.init(s),
d.get(s),
d.put(s), and
d.rem(s).

Init and get affect the referred set only, while put and remove affect the total sub-structure as well.

Initiating a set also means initiating its FIRST, LAST, ACT, PRED and LAST pointers.

Initiating a set (named s) means initiating all the elements that are to belong to the set. SMARN will output

SET s
ELEMENT 1

and the initiation of element number one can start after carriage return is given (denoted <CR>). This takes place by initiating each of the objects the element is defined over. When element one is initiated,

ELEMENT 2

is output. The same procedure is run through for this element. The process may be interrupted by typing 'i' before <CR> (carriage return) and ended by typing 'e'.

ELEMENT 7i<CR>
..
ELEMENT 13e<CR>

After typing 'e', the set is defined.

To load a set means to load all the elements with their objects. If a set is initiated under one of these objects, this will not be loaded.

5.6.9 Projection

Syntax

<projection>::=<set factor>.(<domain>)/<set factor>

Example

Let

role(person) man, woman;
domain(man, woman)couple;
set(couple)married;

be given. The set of all married men is

married.(man)

5.6.10 Selection

Syntax

<selection>::=<projection>where<Boolean expression>/
<projection>

Example

Declarations as in 5.6.7. Persons in s aged 18 years or more are given by

s where person.age > 18

5.6.11 Cartesian factor

<cartesian factor>::=<set term>/
<cartesian factor>+<set term>/
<cartesian factor>-<set term>/
(<element identifier list>)

5.7 Element in set

5.7.1 Element type

Syntax

<element type>::=eref (<domain>)

Semantics

A member of a set is called an element. An element may be a member of one or more sets or it may be detached. It will be defined over one or more object types given by the domain qualification. If identical object types occur in an element, they must be given a role specification, indicating the role of the object in the element.

The element type may by used to declare pointers to elements of specific interest.

Example

eref(d) element

is a reference to elements from domain d.

5.7.2 Element expression

Syntax

<element expression>::=
<simple element expression>/
<if clause><simple element expression>
else <element expression>

<simple element expression>::=<set factor>.
<set manipulator>/
<element variable>/none/<function

designator>/<element generator>/
(<element expression>)

Semantics

The type of an element expression is given by its domain qualification. The value is a reference either to a specific element or none.

If E1 is a simple element expression and E2 is an element expression then

if b then E1 else E2

is an element expression. E1 and E2 must have compatible qualifications, which will be the expression qualification.

By means of the set manipulators FIRST, LAST, ACT, PREV, PRED, SUC and NEXT, one may access the elements in a set, one after the other, to process them.

5.7.3 Element identifier list

<element identifier list>::=<element identifier list>,
<element identifier>/
<element identifier>/
ø

5.7.4 Element generator

<element generator>::=enew<domain identifier>

5.8 Domain

Syntax

<domain declaration>::=domain(<domain>)<identifier list>

<domain>::=<domain>,<role identifier>/<role identifier>/
<domain>,<object identifier>/

<object identifier>

Semantics

A domain is a collection (list) of one or more objects qualifying elements and sets. There is no ordering of the objects, which will be referenced by their names. If there are several identical objects in a domain, they must be separated by a role specification.

SMARN objects, and these only, may be used to define domains.

5.9 Role declaration

Syntax

<role declaration>::=role(<object identifier>)
<identifier list>

Semantics

A role specification is needed only in the declaration of a domain when two or more objects of the same type are to be used. Its role is to give unique access to the various objects in an element.

BIBLIOGRAFY

Allen, Layman (1980); Language, Law and Logic: Plain Legal Drafting for the Electronic Age, in Niblett (1980)

Belkin, J., Blumstein, A. and Glass, W.(1972); JUSSIM, An Interactive Computer Program for Analysis of Criminal Justice Systems. Carnegie-MelIon University.

Bing, Jon (1975); Fra problem til resultat, Jussens Venner 1/1975, Universitetsforlaget, Oslo

Bing, Jon and Harvold, Trygve (1977); Legal Decisions and Information Systems, Universitetsforlaget, Oslo

Byrgesen, J. G. (1975); SISYFOS - A Law Preparation Game, IBM Consumer Executive Seminar on Social Security, Brussels

Bratholm, A. and Hov, Jo (1973); Sivil rettergang, Universitetsforlaget, Oslo

Castaneda, H.-N. (1981); The Paradoxes of Deontic Logic: The Simplest Solution to all of them in One Fell Swoop, in Hilpinen (1981)

Chen, P.P.S (1976); The Entity-Relationship Model, Toward a Unified View of Data, ACM Transactions on Databse Systems 1, No. 1, march 1976

Dahl, Myhrhaug and Nygaard (1982); SIMULA67 Common Base Language, Norwegian Computing Center, Report no. 725

Date, C. J. (1981); An Introduction to Database Systems, Addison-Wesley, Reading, Massachusetts

Dini et al. (1983); Un modello automatico per analisi dei normativi: una proposta sperimentale, in Proceedings of L'informatica guiridica e le communita nazionali ed internazionali, Rome, May 9th-14th

Dijkstra, Edsger W. (1976); A discipline of programming, Prentice Hall, London

Dotterwich, Franzen and Weihmuller(1978); Simulationsmethoden in der gerichtlichen Verfarensforschung, Arbeitspapiere Rechtsinformatik, Heft 12, Institute for Legal Dataprocessing, GMD, Bonn

Eckhoff and Sundby (1975); Rettssystemer, Universitets-forlaget, Oslo.

Franzen, Richard (1983); An Interactive Simulation Model for Analysing Queuing Problems in Alternative Organisational Structures of Civil Procedures, Journal of Law and Information Science, New South Wales Institute of Technology, Vol. 1, no. 3.

Hansen, Johs (1981a); Et EDB-system for analyse av rettslige avgjørelser, CompLex 1/81, Universitetsforlaget, Oslo

Hansen, Johs (1981b); Utvidelse av SARA med flerverdiresultat-er og skissering av et verktøy for modellering av normer, in Hansen, Johs (ed.) , Notater om Deontiske Systemer, CompLex 2/81

Hansen, Johs (1982); Working document(SI),SMARN - A Language for Modelling Legal Norms, SMARN grammar.

Hansen, Johs (1983a); Working document(SII(2)), SMARN grammar. Version II(2).

Hansen, Johs (1983b); SARA- A system for Analysing Discretionary Decisions, in proceedings from 10th International Congress on Cybernetics, Symposium VI, Association Internationale de Cybernetique, Place Andre Rijckmans, B-5000 Namur, Belgique

Hansen, Johs (1984a); Working document(SIII(1)), SMARN grammar. Version III(1).

Hansen, Johs (1984b); Working document(SIII(2)), SMARN grammar. Version III(2).

Hansen, Johs (ed.) (1985); Modelling Knowledge, Action, Logic and Norms, CompLex 8/85, Universitetsforlaget, Oslo

Hilpinen, Risto (ed.) (1971); Deontic Logic, Introductory and Systematic Readings, D. Reidel Publishing Company, Dordrecht, Holland.

Hilpinen, Risto (ed.) (1981); New Studies in Deontic Logic, D. Reidel Publishing Company, Dordrecht, Holland

Hohfeld, W.N. (1923); Fundamental Legal Conceptions, 3. edition, N. Haven 1964

Lawlor, Reed (1980); Computer Analysis of Judicial Decisions, in Niblett (1980)

Malt, Gert Fredrik (1983), Deontic probability, in Hansen (1985)

McCarty (1977); Reflections on TAXMAN: An Experiment in Artificial Intelligence and Legal Reasoning, Harward Law Rewiew 90: 837-893.

McCarty, Shridharan and Sangster (1979); The Implementation of TAXMAN II: An Experiment in Artificial Intelligence and Legal Reasoning, Technical Report LRP-TR-2, Laboratory for Computer Science Research, Rutgers University.

McCarty (1980a); Some Requirements for a Computer-Based Legal Consultant, in Proceedings of the First Annual National Conference on Artificial Intelligence, Stanford University.

McCarty and Shridharan (1980b); The Representation of an Evolving System of Legal Concepts: I. Logical Templates, in Proceedings of the Third Biennal Conference of the Canadian Society for Computational Studies of Intelligence. Victoria, British Columbia.

McCarty and Shridharan (1980c); The Representation of Conceptual Structures in TAXMANII. Part One: Logical Templates. Technical Report LRP-TR-4, Laboratory for Computer Science Research, Rutgers University.

McCarty and Shridharan (1980d); The TAXMAN Project Towards a Cognitive Theory of Legal Argument, in Niblett (1980)

McCarty, L. Thorne (1981); The Applications of Artificial Intelligence to Law: A Survey of Six Current Projects, Proceedings of the AFIPS National Computer Conference, Chicago,Illinois, May 4th-7th.

McCarty L.T and Shridharan, N. S., (1981); The Representation of an Evolving System of Legal Concepts: II Prototypes and deformations, in Proceedings IJCAI-81, University of British Columbia, August 1981, 246-53

McCarty, L Thorne (1983); Permissions and Obligations, Proceedings of the 8th International Joint Conference on Artificial Intelligence, also in Hansen (1985)

McCarty, L Thorne (1985); A Computational Deontic Logic, in Proceedings of Logica, Informatico et Diritto, Firenze

Naur, Peter (1964); Revised Report on the Algoritmic Language Algol60, A/S Regnecentralen, Copenhagen.

Neal, Alan (1984); The Leicester project, presented to the Warwick Conference, held by the Society for Computers and Law

Niblett, Bryan (ed.) (1980); Computer Science and Law; Cambridge University Press, Cambridge

Nijssen, G. M (1981); An Architecture for Knowledgbase Software, paper presented to the Australian Computer Society, July 30. 1981

Pørn, I. (1970); The Logic of Power, Blackwell, Oxford

Ross, Alf (1953); Om ret og retfærdighed, Nyt Nordisk Forlag, Arnold Busch, København

Rynning, Paul (1976); En fremstilling og vurdering av reglene for tildeling av bostøtte, Skriftserien JUS og EDB nr 17.

Sergot, Marek (1982); Prospects for Representing the Law as Logic Programs, in K. L. Clark and S.-A. Tærnlund, Academic Press, London

Stamper, Ronald(1980); LEGOL: Modelling Legal rules by Computer, in Niblett (1980)

Sundby, Nils Kristian (1974): Om Normer, Universitetsforlaget, Oslo.

Svoboda, W.R. (1985); The Use of Models in Planning Legislation, Proceedings of Logica, Informatica, Diritto, Firenze

Tidligere utgitt i skriftserien CompLex

Alle heftene i skriftserien CompLex kan kjøpes fra Universitetsforlaget eller fra Norsk forening for jus og edb. Postboks 7557 Skillebekk, 0205 OSLO 2. Her kan man også tegne abonnement eller abonnere på "vaskesedler" som gir en kort omtale av hver rapport etterhvert som de kommer ut, og som inneholder bestillingsseddel. Abonnement på "vaskesedler" er gratis.

CompLex 1/81
Johs Hansen
Et edb-system for analyse av rettslige avgjørelser

CompLex 2/81
Johs Hansen (red)
Notater om deontiske systemer

CompLex 3/81
Vidar Sørensen
Informasjonssystem mm for Norges Geotekniske Institutt

CompLex 4/81
Kjetil Johnsen
Systemtekniske konsekvenser av persondatalovgivningen

CompLex 5/81
Marit Thorvaldsen
Teledata og rettsinformasjon

CompLex 6/81
Anne Kirsti Brække
Postmonopolet

Complex 7/81
Knut S Selmer (ed)
The LAWDATA Papers

CompLex 1/82
Norsk forening for jus og edb & Svenska föreningen för adb och juridik
Nordiske personregisterlover, Council of Europe Convention og OECD Guidelines

CompLex 2/82
Harald Brænd og Vidar Sørensen
Oljevern: Brukerbehov og kildemateriale

CompLex 3/82
Jon Bing og Dag Frøystad
Rettskildebruk

CompLex 4/82
Thomas Prebensen Steen
Post- og televerkets regulerte ansvar

CompLex 5/82
Datatilsynet
Årsmelding 1981

CompLex 6/82
Birger Eckhoff
in cooperation with Jon Bing, Dag Frøystad and Anja Oskamp
CATTS: Computerized program for teaching text retrieval systems

CompLex 7/82
Jon Bing
Informasjonssystemer for Trygderettens kjennelser

CompLex 8/82
de Mulder, Oskamp, van der Heyden and Gubby
Sentencing by Computer: An Experiment

CompLex 9/82
Colin Tapper
An Experiment in the Use of Citation Vectors in the Area of Legal Data

CompLex 1/83
Arve Føyen
Utredning om endringer i personregisterloven

CompLex 2/83
Stein Schjølberg
Computers and Penal Legislation

CompLex 3/83
John S Gulbrandsen og Terje Hoffmann
Rettigheter i idrettsarrangementer

CompLex 4/83
Sally Moon and Anja Oskamp
The Law of Legal Information Systems: Two Essays

CompLex 5/83
Datatilsynet
Årsmelding 1982

CompLex 6/83
Olav Torvund
Betalingsformidling

CompLex 7/83
Else Ryen
Lov og lovmottaker

CompLex 8/83
Daniel Stripinis
Probability Theory and Circumstantial Evidence

CompLex 9/83
Frede Cappelen
Edb-basert informasjonssystem for forvaltningens praksis

CompLex 10/83
NORDIPRO
Legal Acceptance of International Trade Data

CompLex 11/83
Jørgen Hafstad og Thomas Prebensen Steen
Teleksrett og merverdiavgift på programvare

CompLex 12/83
Kristin Kjelland-Mørdre
Om forenkling av regler

CompLex 13/83
Jon Bing
Edb: Mulighet og problem ved forenklig av regelverk

CompLex 14/83
Tarjei Stensaasen (red)
Utvalgte emner i jus og edb (3. utgave)

CompLex 15/83
Anette Klafstad og Ulf Alex Samer
Plan for et forsikringsrettslig informasjonssystem

CompLex 1/84
Jon Erling Skjørshammer
Kabelnett: Bygnings- og ekspropriasjonslov

CompLex 2/84
Tore Andreas Hauglie og Dag Wiese Schartum
Forlsag til et helserettslig informasjonssystem

CompLex 3/84
Justisdepartementet
Den elektroniske grunnbok

CompLex 4/84
Datatilsynet
Årsmelding 1983

CompLex 5/84
Tove Fjeldvig
Tekstsøking: Teori. metoder og systemer

CompLex 6/84
Jon Bing
Offentlighetsloven og edb

CompLex 7/84
Gunnar Bach. Beate Jacobsen and Vidar Stensland
The National Social Insurance System of Norway

CompLex 8/84
Dag Frøystad
Data Protection in Practice I: Identifying and Matching Elements

CompLex 9/84
Tove Fjeldvig og Anne Golden
Automatisk rotlemmatisering -et lingvistisk hjempemiddel for tekstsøking

CompLex 10/84
Elling Øyehaug Osc
Retstidene som informasjonssystem

CompLex 1/85
Jon Bing
Data Protection in Practice II:
International Service Bueraux and Transnational Data Flows

CompLex 2/85
Jon Bing
Opphavsrett og edb

CompLex 3/85
Dag Wiese Schartum
Codex, Calculation and Computers

CompLex 4/85
Olav Torvund
To informasjonsrettslige arbeider

CompLex 5/85
Datatilsynet
Årsmelding 1984

CompLex 6/85
Hans Chr Aakre (red)
Utvalgte artikler i rettsinformatikk

CompLex 7/85
Johannes Hansen
SARA: Brukerveiledning og programdokumentasjon

CompLex 8/85
Johannes Hansen (ed)
Modelling Knowledge, Action, Logic and Norms

CompLex 9/85
Thomas Prebensen Steen (red)
Kompendium i informasjonsrett

CompLex 10/85
Tarjei Stensaasen
Opphavsrettslovens § 43 ("katalogregelen")

CompLex 11/85
Jon Bing
Straffelovens definisjon av "trykt skrift" anvendt på datamaskinbaserte informasjonssystemer

CompLex 12/85
Magnus Stray Vyrje
Vanhjemmel, opphavsrett og datamaskinprogrammer

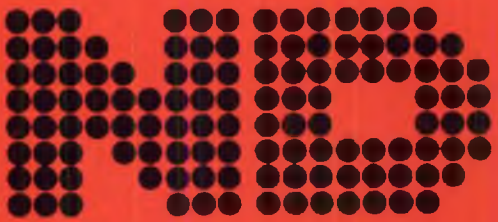
CompLex 1/86
Anne Kirsti Brække
Formidlingsplikt for kabeleier

CompLex 2/86
Dag Wiese Schartum
Stans av edb-tjenester i krigs og krisesituasjoner

CompLex 3/86
Ingvild Mestad
"Elektroniske spor" - nye perspektiv på personvernet

CompLex 4/86
Jon Skjørshammer
Opphavsrett, databaser og datamaskinprogrammer: Kontraktsrettslige aspekter

CompLex 5/86
Datatilsynet
Årsmelding 1985



Norsk Data

ISBN 82-00-07833-7

GCS, Oslo