D A T A L O G I

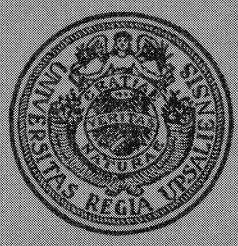
L A B O R A T O R I E T

# Project Technical Report for 1970 and plans for 1971

Erik Sandewall, H-J Holstein
Mats Nordström, Jaak Urmi

UPPSALA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCES



D A T A L O G I

L A B O R A T O R I E T

# Project Technical Report for 1970 and plans for 1971

Erik Sandewall, H-J Holstein

Mats Nordström, Jaak Urmi

Contents:



Sections A through H are individually paginated.

## A. Research objectives

Research at DLU(x) aims to apply methods in artificial intelligence to computer problems that involve intense processing of small data bases.(xx) By a "small" data base, we mean a data base whose size is the order of magnitude of core on a large-size computer. By "intense" processing, we mean processing which has a complex logical structure; which requires complex programs; and where the complexity depends on an inherent complexity in the given task, rather than complexity which was introduced for efficiency reasons when the problem was formulated for the computer. Mechanized logical deduction (= "automatic theorem proving") in a question-answering program is a typical example of intense processing. Natural-language conversation when the computer is to maintain a data base of a few hundreds or thousands of common expressions or constructions is another. DENDRAL-type systems, which attempt to extract through conversation the know-how of a human specialist in a particular field, provide a third example.

We shall shortly refer to this class of problems as "small data base" problems, but the criterium of complex processing is always implicit and essential. Among small data base systems, we do not include data bank type systems, where the size of data is very large, where processing is by necessity very shallow, an where the major problem is to bring data from external storage into, through, and out of the computer as fast as possible. On the other hand, the requirement that there shall be some data base implies that we exclude programs, however complex, where the data represents numerical quantities, or where the data is a string of characters with little or no structure besides the trivial,

---

(x) DLU stands for Datalogilaboratoriet i Uppsala. "Datalogi" is a Scandinavian word for a subset of Computer Science which includes systems programming, data structures, artificial intelligence, etc. but which does not include numerical analysis, information retrieval, or some business data processing.

(xx) The development of conversational programming systems is one major task at DLU but has the standing of a sub-goal. More about this later.

linear one (e.g. text editing programs).

Let us make this more concrete through some examples of application of small data base systems:

1. Question-answering programs. These are programs which maintain a data base of simple facts about common-life situations, and which can accept and answer question in natural language, using the data base plus some common-sense reasoning. Usually, it is required that new facts can be added to the system by natural language input. Question-answering systems have been proposed

1a. as public information terminals at airports, railway stations, department stores, etc.

1b. as public information terminals where anybody can get information about government and other large organizations: how they are organized, which services they can offer, how and when their decisions are made, etc.

1c. for military command and control systems, etc.

A further discussion of this application is in "Artificiell intelligens (1970)", part B.

2. Model management systems, which maintain a data base of relatively many, simple models. Detailed example: in ecological research there is an abundance of circulation models. From a systems point of view, these are closed compartment models which are normally in a state of equilibrium or periodicity. Compartments in the model may represent river - lake - atmosphere - clouds - rain or (in another model) earth - plants - decaying organic matter. In each model there is one or a few substances which circulates, e.g. a poisonous substance. The direction and velocity of the transfers depends on parameters (like pH value or temperature), and such parameters may also be affected by the circulating substances. In an analysis of the models, one is interested in knowing the state of equilibrium, and in the effects of disturbances in the system.

In practice, the models are not independent. Instead, a variable whose value is affected by one model, may control the behavior of another model. This is harmless in analysis of the stationary state, since one can still measure some variables and use the model to compute the others, but it is important in the study of effects of a disturbance, e.g. addition or elimination of a pollutant.

In this situation, it is reasonable to let a program maintain a small data base of models, and make qualitative analysis of the model structure to answer questions like "in which models is the water's oxygen content a parameter?"; "is there any component in the exhaust which alone could be responsible for the following four effects:.... ?", or "is there reason to believe that a change in A could cause a change in B?" If the answer to the last question is Yes, then it still takes numerical computations to determine whether thresholds were passed etc. so that the change in A did indeed cause the change in B. On the other hand, if logical analysis determines that a change in A could not possibly cause a change in B, then we have been able to avoid a lot of computation, and we probably have a more reliable answer than we could get from simulation.

3. Computer-aided instruction. In some advanced CAI systems, one wants the computer to carry on a mixed-initiative conversation with the student about the subject-matter of instruction. This means that the computer should both make questions to the student and answer the student's questions. Such a system requires that the computer maintains a small data base of knowledge about the subject-matter, and it is therefore our third example of applications.

The mere observation that a lot of interesting computing problems involve processing of small data bases, is not sufficient reason to let a research program and research group concentrate on such computing problems. However, we also have a set of methods and ideas which we believe are relevant for most small data base applications. These methods and ideas are outlined in section C ("Research strategy") of this report.

Most applications for small data bases assume intense man-machine inter-

action. Time-sharing computer systems are therefore an essential tool for this research, and practical necessity has forced us to make the development of programming systems for time-sharing one of the major sub-goals for the group. The PILS system (paged interactive LISP system) which we presently develop is an example of this. - Other sub-goals relate more directly to the major source of inspiration, i.e. artificial intelligence research, and are discussed in section C.

In summary, work at DLU attempts to apply artificial intelligence methods to small-data-base problems. We have methods, and ideas for further methods, which we believe are useful for many problems within the given class. These methods and ideas range from the practical (e.g. development of basic software) to the theoretical (e.g. methods of automatic logical deduction or "theorem-proving"). The basic work to develop those ideas is an essential purpose of the group, and goes on in parallell with the work on various applications.

## B. Background; work during 1970

Work at DLU is conducted in two research groups with distinct but converging history:

The research group for "datalogi"(x) was organized in early 1970 after the advent of some fresh research support. Before 1970, several people in the Computer Sciences department (of Uppsala University) had done individual projects in the artificial intelligence field, and one person (Erik Sandewall) had participated in the "Swedish Question-Answering Project" whose other members worked in Stockholm (at FOA P). The research group for "datalogi" was set up within the C.S. department to stabilize these activities. In January, 1971 it has the following members:

Anders Beckman
Mats Cedvall
Lennart Drugge
Anders Haraldson (§)
Fred Jacobson
Otto Lindgren
Mats Nordström (§)
Erik Sandewall (§)
Arne Tengvald
Jaak Urmi (§)
Olle Willén (§)

Persons marked with a paragraph sign participated in projects during 1970, the others entered the group in January, 1971. The following persons participated during 1970 but have now left the group:

Diz Breslaw (January - December)
Lars Nilsson (January - May)
François-Yves Villemin (January - May)

The following persons are associated, and participate in some of the group's activities:

---

(x) see footnote on page A.1

Rodger Knaus
Kalle Mäkilä
Torgny Tholerus

The research group for sociocybernetics originated in the department of sociology, which has been conducting a relatively large-scale project ("Märsta-projektet") since 1960. This project required a lot of computer processing, mostly of a routine character, but there emerged the need and the interest for using computers in less trivial ways, e.g. for model management systems similar to the one that was outlined in section A. The research group for sociocybernetics is now part of the Sociology and the Computer Sciences Departments, and has the following members:

Hans-Jürgen Holstein
René Reboh
Lennart Ståhlberg

The plans for work during the period Jan. 1, 1970 to July 1, 1971 are given in "Artificiell intelligens (1970)" (reference I ), and contains the following main headings:

1. LISP systems, especially (PILS project) development of a paged interactive LISP system for the Siemens 300 series computer.
2. Question-answering systems (including deduction problems and natural-language processing problems)
3. INTERFORM project: Data base oriented formula manipulation (see also section G of this report)
4. ATMAN and ECOSYS projects: Model analysis in sociological models (see section H)

It was presumed that programming in the last three projects should be done in LISP and run on other LISP systems before the PILS system became available.

Let us consider each of those four headings and compare plans a year ago with the actual results during 1970. It must be understood, then, that since the very nature of research is discovery, problems under

attack often turn out to be harder, or easier than expected; some problems turn out to be irrelevant, and new interesting problems come up. Therefore, it must never be a purpose in itself to conform to plans. On the other hand, if we change plans we should at least know what we are doing. It is also interesting to see whether the quantity of results agrees with what was predicted.

## 1. LISP systems

Work under this heading is supported mainly by FOA and UDAC. Plans a year ago called for programming of an interactive LISP system (the PILS system) for the Siemens 300 series computers. A computer was expected to arrive in February. The work to write the interpreter and to transfer a compiler and some user service programs (editor, DWIM package, advise package) was to start immediately and to be finished by July 1, 1971.

LISP F. The Siemens computer was delayed in several ways and the installation was not available for users until the beginning of October. In order not to delay various projects which presume access to an interactive LISP, we then decided to write a LISP interpreter in FORTRAN for use on commersially available, RAX-like, time-sharing systems. Such an interpreter must by necessity be relatively slow, and since we are confined to a rather limited core size and can not use secondary storage, the LISP user does not get very many LISP cells. However, the system is useful for debugging and tutorial purposes.

Work on the FORTRAN interpreter for LISP (called LISP F) started in January, and the system was operative for users in March and officially debugged in September. It is available on the DATEMA time-sharing service, and is used regularly. The user presently gets 9000 LISP cells. The preliminary version of the documentation was issued in October; the definite, sligthly updated version is being typed. This work was performed by Mats Nordström, with some assistance of Erik Sandewall.

PILS. The present (January 25) status of the PILS system is as follows: the interpreter and a number of basic functions are running, and obvious bugs have been removed so that simple test examples come through. The

reportoire of available functions includes elementary list functions prog, and I/O. Paging routines have not been written, so the user is confined to core. (The system is organized so that the paging routine can be added without reprogramming any of the existing code). Garbage collection does not work yet, but is expected to work any day. In summary, the system is coming to life. A two-paged users' instruction has been written, and some selected users have tried to run their LISP programs on the PILS system, with varying results.

Work on PILS has the following history: Jaak Urmi visited BBN (Bolt, Beranek, and Newman, Inc., in Cambridge, Mass.) from April to August, to learn about their LISP systems for SDS 940 and PDP-10, from which the PILS system has been modeled. System design for PILS was done in August; September was spent programming, and debugging could start in October when the computer arrived. This work has been done by Jaak Urmi, with assistance of Anders Haraldson (October - now) and Diz Breslaw (September - December). - Expectations are that the original intention to have the paged system, compiler, and user service programs running by July 1 will succeed.

Besides these major efforts, some smaller systems projects have been performed, namely:

LISP editing and tracing systems. A small interactive list structure editor for use on programs and data in LISP systems, and a simple trace package, were written in LISP by Diz Breslaw. The packages are intended for use in LISP F and other small, interactive LISP systems. Work started in April and was completed in September. Documentation is in the LISP F1 manual.

Modification of the 3600 LISP interpreter. The existing interpreter for 3600 LISP has been given a careful overhaul. The system has been speeded up by 50 - 70 %. A number of new standard functions and facilities have been added, in particular, error handling routines (so that the user can define a function which is to be called during interpretation if a function symbol or variable is undefined, thus avoiding an error interrupt and a LAP facility (so that a user or a LISP program can provide an

assembly language program, which is assembled and linked with the LISP interpreter). Work on LAP has been performed by Kalle Mäkilä, all other work by Jaak Urmi. The work was completed in April. Documentation is available in {Urmi 1970}.

TT-LISP (a compiler for a modified LISP, for use on the CDC 3600). This work has been performed by Torgny Tholerus. The system is running but not (yet) documentated.

## 2. Qestion-answering systems

Work under this heading is supported mainly by NFR and FOA. Plans as formulated in "Artificial Intelligence (1970)" expected the SQAP project (see below) to be terminated during the third quarter of 1970; to use the PILS system before July 1, 1971 for experiments with a number of small and medium size programs in various fields, including natural-language processing, and to do theoretical work(x). In a grant application to NFR for the period October 1, 1970 to July 1, 1971 this was further specified to address the following tasks:

(a) Transfer some existing programs to available, interactive programming systems, so that the user (and the programmers that will be involved in future projects) will get the feeling for computer conversations.

(b) Modify those programs so that the user can "teach" the program new phrases as the need arises.

(c) Start work on the problem of making programs "understand" continuous text, where the meaning of sentences and the relationships between the sentences are often more complex than the presently assumed, simple categories of assertion - question - instruction.

Clearly, (c) is the major task.

SQAP project. The SQAP project (Swedish Question-Answering Project) started in early 1968, and is performed in cooperation between FOA and DLU. The major programming effort is at FOA, and is reported for elsewhere. The work in Uppsala is to contribute some of the methods for the system. Goals for the project and work before 1970 is reported in {Palme 1970} and in "Artificiell intelligens (1970)".

(x) "primarily on heuristic search methods", but not excluding problems in natural language processing.

Plans a year ago expected the project to be completed during 1970. This did not happen, because some problems were harder than we thought, and because most people in the project were partly occupied with other tasks (planning a time-sharing computer facility at FOA).

The data base in the SQAP project is an SPB structure, as described in {Sandewall 1969}. During 1970, the data structure has been extended through the introduction of a variable-like concept, and the deduction and retrieval procedures have been extended to accomodate the new data structures. This work has been performed by Erik Sandewall and is presently not documented. Moreover, Anders Haraldson and Lars Nilsson have written and debugged small programs for experiments with various search methods. One of these programs is described in (Haraldson 1970). Most of this work was done during the first half of 1970.

Preliminary studies for SQAP 2. Using the experience from the SQAP project, we have studied some possible alternative approaches which might be used in a future SQAP 2 project. In particular, we have developed:

(1) An approach to the problem of computer understanding of continuous text. This is the (c) problem mentioned in the NFR application (page B.5). Our approach, which has not been documented, makes rather strong assumptions on internal representation and retrieval in the question-answering system, which caused us to attempt the development of

(2) A new, internal, representation of conceptual (= natural-language) information, which relies more heavily on predicate calculus notation, and therefore is called PCF, for "predicate calculus formulation" (of conceptual information). The new notation promises to be more flexible than the notation used in the SQAP project, but it may instead make it harder to write efficient retrieval procedures for the data base. PCF is described in {Sandewall 1970a} and {Sandewall 1970b}. Sample translations of some Swedish sentences (from a children's book) are given in {Sandewall 1971a}.

(3) System design for a predicate calculus data base (PCDB) management system. The PCDB package would be responsible for storing and retrieving predicate calculus formulas in a small data base, and for compiling axioms into relatively efficient LISP programs. This is

an attempt to handle, or at least see if we can hope to handle, the increased complexity of retrieval if the PCF representation is used, but we believe that the package will have several other applications as well. Important parts of the program for the PCDB package have been written but not yet debugged. Some of the work to date is described in the PCDB Users' Manual (the first few chapters available), and in the PCDB/n preliminary program documentation. The work to set up and debug this package on the PILS system is just being started.

(4) A LISP function package which generates and simplifies LISP function definitions. The package is called REDFUN. This work does not logically fall in the field of natural-language processing, but it is an essential tool for PCDB, which is essentially a function generator. REDFUN has the same status as PCDB, i.e. important parts of the program have been written but not yet debugged. A preliminary program documentation is avilable.

The REDFUN package uses LISP's FUNARG feature, and this raised some implementation problems of general interest. A proposed solution is given in (Sandewall 1970c).

Work under the SQAP 2 heading has been done by Erik Sandewall. The PCDB and REDFUN packages are part of the PILS experiments that were postulated in "Artificiall Intelligens (1970)".

QAMEAS. A question-answering program for measurements has been written in LISP by Lennart Svensson as a term project (trebetygsuppgift). The program can accept statements like "1 foot is 3 decimeters" and answer questions like "how many feet per second is 40 kilometers per hour?". It was developed in batch mode for 3600 LISP during April to August, 1970, and was later transferred to the interactive LISP F system by Anders Haraldson. A program documentation is available in one copy and can be Xeroxed. This is the only work that has yet been performed on NFR problems (a) above. QAMEAS and several other programs will later be transferred to the PILS system.

Two further term projects (trebetygsuppgifter) in the C.S department during 1970 relate to natural-language processing and shall be briefly mentioned:

FTNPARS, a FORTRAN program for an arbitrary context-free grammar. This work is performed by Lennart Drugge, who is now a member of the research group for "datalogi". The work is expected to be finished in February.

Categorial parser. A parsing program for a categorial grammar was written in ALGOL by Norbert Schüller. The program is running but not yet documented.

## 3. INTERFORM project

Work under this heading was supported mainly by STU. A description of the project, results during 1970 and plans for 1971 and 1972 are given in section G. Shortly, the project started in the spring of 1970 with a survey of existing litterature. During the autumn, the structure of the program and the data has been decided, and some program modules have been written and debugged. The project is running on schedule and is expected to be completed in the latter half of 1972.

## 4. ATMAN and ECOSYS projects

Work under this heading is supported by RJF and conducted within the research group for sociocybernetics. During 1970, this group has been working on explorative studies as well as on computer programs in the three main areas of sociocybernetics:

a. Methodology: Continued development of an interpreter for a language developed especially for simple and flexible description of standard processing of sociological data (continued from 1968/69).

b. Microsociology/psychology: Explorative studies of artificial attitu systems, especially an attempt at programming a model of theory formation.

c. Macrosociology/ecology: Some simple simulations of abstract ecological systems. The major effort is however on the design of simulation models that will be used during 1971/72.

This work has been performed by Hans-Jürgen Holstein, with the assistance of René Reboh and Lennart Ståhlberg.

This terminates the discussion of the various branches of research. We conclude that considerable progress is being made in each of these branches.

The volume of work has grown considerably during 1970, and through the formation of the research group for "datalogi" it has become more closely integrated. We have therefore had reason to reconsider and reformulate the objectives and the strategy for this work. The research objectives have been described in section A of the present report; they are to apply methods in artificial intelligence to computer problems that involve intense processing of small data bases. We did not find this formulation of the group's objectives until in the autumn, although the basic ideas were implicit in the research proposals that were made earlier in the year. The strategy which has been selected for work with these objectives, will be described in the next section.

References

1. Erik Sandewall, Jaak Urmi m.fl.
Artificiell Intelligens (1970) (in Swedish)
January, 1970

2. Anders Haraldson
Ett LISP-program som överför PROPLAN-notation till intern SPB-notation (in Swedish)
March, 1970

3. Mats Nordström et al.
LISP F1: A LISP Interpreter Written in FORTRAN
February, 1971

4. Jacob Palme
Making Computers Understand Natural Language
FOA C-rapport, October 1970

5. Erik Sandewall (1969)
A Set-Oriented, Property-Structure Representation for Binary Relations, SPB
in Machine Intelligence 5 (Meltzer & Michie, eds.)

6. Erik Sandewall (1970a)
Representing Natural-Language Information in Predicate Calculus
in Machine Intelligence 6 (Meltzer & Michie, eds.)

7. Erik Sandewall (1970b)
Formal Methods in the Design of Question-Answering Systems
in Artificial Intelligence, vol. 2 (1971)

8. Erik Sandewall (1970c)
A Proposed Solution to the FUNARG Problem
CS Dept. report nr 29 (Nov. 1970)

9. Erik Sandewall (1970d)
PCDB User's Manual
First version, Dec. 1970

10. Erik Sandewall (1970e)
PCDB/n program documentation
December, 1970

11. Erik Sandewall (1970f)
REDFUN program documentation
December, 1970

12. Erik Sandewall (1971a)
Enkel PCF för svenska (in Swedish)
January, 1971

13. Jaak Urmi (1971)
3600 LISP U3 User's Manual
September 1970

All references are available through Datalogilaboratoriet (address: Sysslomansgatan 25, 752 23 Uppsala, Sweden)

## C. Research strategy

When formulating a research strategy, we made some basic assumptions:

(a) Work on applications projects should be performed in parallell with basic research;

(b) The basic research in this field should not merely aim at the development of methods which may be useful in som applications, but also at programming these methods into subroutines which can be used in the programs for several applications. We do not believe that a universal program for handling small-data-base type problems is possible or desirable, but we do believe in building a "toolbox" of programs and methods.

(c) In most of these projects, it is absolutely necessary to have access to a highly interactive time-sharing computer facility. Since a "small" data base can not be expected to be minute, this time-sharing system must be based on a virtual memory and a swapping mechanism.

The second assumption implies that we need a common programming language within the group. Since we wish to bring in existing programs from outside, this language must be reasonably wide-spread. By the nature of our tasks, we need a language with good list processing facilities, whereas facilities for efficient numerical computations are of minor importance. These considerations leave us a choice between the following languages:

ALGOL 68
LISP
PL/I
SIMULA 67

Among these, LISP has an advantage in that so many programs in the artificial intelligence field have been written in LISP. The others have the advantage of being more general-purpose languages, and PL/I also has

the advantage of being more wide-spread and more generally available.

In the final decision, two more things had to be taken into consideration First, we were given free and abundant access to a small/medium-size computer, a Siemens 305 with 16K 24-bit words,a disk which could be used as swapping memory (although it would be relatively slow), and so on. None of the above languages were available on this computer, so we would have to implement the language ourselves. We did not want to put more work than necessary into this system programming. This was a strong argument for LISP, since it is well-known how LISP can be implemented using boot-strapping techniques.

The other consideration was that we expect programs in our class of applications to become relatively large. We therefore want computer support for maintaining and analyzing our programs. Such support can be given by programs like Warren Teitelman's DWIM (Do-What-I-Mean) package (compare "Artificiell Intelligens (1970)"). This means that we need a language with a good (easy-to-analyze) program structure, a good data structure, and an established method for representing programs in the format of this data structure. This property is characteristic of interpretation-oriented languages, including LISP, but the other three languages above do not have it.

For these reasons, we decided to use LISP as the common language in our projects. (Admittedly, some less rational factors also played a part in the decision).

With some simplification, we can therefore say that work at DLU is performed on three levels:

(a) Special-purpose systems programming, aiming at the development of an interactive LISP system, which is to be used as a tool for the various projects. See section D of this report.

(b) Development of subroutines, auxiliary programs and methods which ar useful for complete programs or for other subroutines. One such auxiliary program is described in section E.

(c) Work on complete programs which have been designed for the benefit of a user, rather than for a programmer. This includes both programs for applications, such as the INTERFORM project (see section F. of this report), and programs for use in our own research, such as the experiments with models of natural-language discourse (see section F.).

The border-line between the levels is very vague, so this has the character of a spectrum, rather than three distinct floors. For example, under "systems programming" we include both the machine-code work of writing an interpreter, and the work of transferring an existing LISP compiler, structure editing package, DWIM package, M-notation input package, etc. to this system for bootstrapping. Similarly, code which was written as part of one special-purpose, complete program has turned out to be useful as "tools" in other programs as well. In spite of this, the description of three levels has a value as an organizational guide-line.

On the toolbox level, our strategy is to bring in tools from abroad or develop them ourselves (whichever seems more adequate in each case) at the time when we know we need the tool. It would be stupid to put a lot of work into a tool-box before we knew when and how we could use it. In accordance with this, we like to define applications projects so that the development of some general-purpose tool becomes a necessary part of that project. In order to facilitate planning in this respect, we have prepared a list of tools which are desirable, and which will be brought in or developed as the need arises. This list is given in an appendix. It must be understood that the list is dynamic, and that we update it every few weeks.

The balance between bringing in a program from abroad or developing it here requires some comment. In most cases, it cost us much less work to bring in a suitable program from abroad, than to develop it ourselves. Development of tool-box programs here is only meaningful when we have formulated our requirements on a tool, and found that there is none available. On the other hand, it should be stressed that the task of bringing in outside programs, although less troublesome, is not trouble-

free. Programs of this character are never "closed"; you always have to go into them and change something. This means that we need to have somebody here who understands the workings of the program. Usually, the best way for him to find out, is to go where the program was written, for a few weeks. - This explains why the budgets in our project proposals normally assign a relatively large amount to "travelling".

In summary, the research strategy is to do work on three levels: systems programming for LISP; toolbox development with our own and foreign programs; and applications projects.

This terminates the description of the general research strategy. The remaining sections of this report will describe the plans for each individual project at DLU.

## D. Programming systems

Work during 1971 will be dominated by the PILS system. The LISP F system described in section B is considered as completed, and we will only do minor improvements as we need them.

Let us first repeat some of the background for the PILS project. PILS stands for "Paged Interactive LISP System". The decision to start this project was made in January 1970. For this purpose Jaak Urmi visited BBN (Bolt Beranek and Newman Inc., Cambridge, Mass.), who were judged to have the best LISP-system existing. He stayed at BBN from April to August to learn their system.

Work on systems design and implementation decisions for PILS started in August, and programming could start on a small scale in September. Due to various reasons (mainly delay in the delivery time of the Siemens-305 computer) debugging and tests could not start until October.

It was decided to define the PILS project as consisting of 4 phases as follows

Phase 1 PILS-1
Programming of a basic interpreter with all basic functions. This version of the system should not be paged (but should be prepared for it), and should support one user. The system is designed to run under the standard Siemens Operating system ORG 1.

This system is called PILS-1

Phase 11 PILS-2
Extending PILS-1 to include paging in a 21 bit address-space. The extendid version is called PILS-2. PILS-2 is intended to support one user.

Phase 111 PILS-3
Writing a special purpose operating system to support PILS-2 and allow timesharing with up to 4 users. The limit 4 users is set out of various reasons, the most important being performance considerations and a maximum

of 4 teletype I/O-channels on the Siemens. More teletypes could possibly be attached on the A/D-channels. This version has the name PILS-3

Phase IV PILS-4

Writing a compiler in LISP and bootstrapping. This phase is implementing all kinds of system library functions, structure editor, program correcting, interphases between user and sytem and all kinds of "goodies" as described in "Artificial Intelligence (1970)".

This is the final step in PILS.

## Time estimates

The following time estimates were made in August, 1970:

PILS-1 should be finished 1970

PILS-2 should start when PILS-1 was finished and should be finished about March 15, 1971

PILS-3 was to start when the needs and demands on an OS became clear and the weak points in ORG appeared

Work on PILS-3 was supposed to run in parallell with PILS-2 and be finished no later than the middle of May, 1971

PILS-4 was supposed to run in parallell with the other phases with a growing use of the machine the further the other phases got. PILS-4 should be finished by July 1971.

## Present State and Plans until July 1, 1971

PILS-1 Due to the delay in the delivery of the computer and the documentation for it, something of a state of emergency appeared for PILS-1. Thanks to hard work made by the group working at the project we managed to keep the deadline.

PILS-1 is today completely finished, and available for users (though it is quite troublesome to use it). At some points it surpasses the goals set up (e.g. error handling).

The people who have worked on PILS-1 are

Diz Breslaw who wrote a macro-expander in LISP and most of the macros used in PILS-1. The macro expander was never used but the macros gave a clear hint on what we needed. (We also plan to use it in the future for some other purposes).

Anders Haraldson who simulated the logic of the central interpreter in LISP and did some programming.

Jaak Urmi who supervised the work and did most of the programming and debugging

PILS-2 Jaak Urmi has started work on the paging routines, and most of the program is written but not yet tested.

PILS-2 is expected to be finished ahead of schedule. (February 15 - March 1).

PILS-3 Design work on the OS was stared in the beginning of January. The programming is expected to start before the end of January. The work is done by Fred Jacobson. One more person will possibly be assigned to the programming as Fred Jacobson will not be able to devote much time for the project.

PILS-4 This part is running slowly. The present effort is concentrated on study of a structure editor by Rodger Knaus. Most functions which will be bootstrapped into the system are available from BBN. Under this part Jaak Urmi is planning to visit BBN for a few weeks to get a detailed knowledge of the techniques used in a LISP compiler and to learn the bootstrapping process.

Conclusions. All parts of the PILS project have been successful so far, and all deadlines have been kept. There are no reasons to believe that there will be any significant delays in the remaining phases.

The know-how gathered about LISP systems at DLU has already started to pay off. UMDAC (Umeå computing center) has shown interest in PILS and plan to start an equivalent project with DLU as consultant.

Plans for 1971/72

In december Jaak Urmi proposed a fifth step in PILS called PILS-5. This step is systems research oriented by its nature. It involves such topics as optimizing the page turning routine, using knowledge about how LISP works. PILS-5 also includes theoretical and practical work on the lines indicated in a report under preparation by Erik Sandewall. The basic ideas are to let a user have a back-ground job which operates on the same data as the foreground conversation. The background job might be initiated by another job or by the user. The system should also keep a history of the computation process to facilitate restarts etc.

## E. Management of small data bases

Since work at DLU centers on applications of small data bases, we would like to have standard programs which facilitate storage and retrieval in such data bases. Let us first specify our requirements on such a program; then discuss how some existing programs meet the requirements; and finally describe a project which is performed at DLU, namely the PCDB (Predicate Calculus Data Base) project.

We have the following requirements on a standard program for management of small data bases:

(a) It should be possible to use it in a LISP computing environment (since we use LISP as the common programming system)

(b) It should provide a more problem-oriented and less computer-oriented language for expressing data (or facts, if you please) and retrieval requests (questions). It must be significantly more problem-oriented than LISP's built-in data structure, i.e. "S-expressions stored on property-lists".

(c) At the same time, the program's requirements on storage space and retrieval time must be comparable to those for LISP property-lists, or else the program will never be used.

(d) The primary purpose of the program must be to assist or supplant the user in writing search routines for retrieval. A program which stores data for the user, but leaves him on his own when it comes to search and retrieval, is not very useful. This immediately implies that the program must be able to accept not only elementary facts, but also inference rules, i.e. descriptions of those logical properties of the data that form the basis for the retrieval routines.

Let us illustrate the (b) and (d) criteria in the context of predicate calculus, which is one candidate (although certainly not the only candidate) for a problem-oriented representation. Suppose we want to perform that standard exercise: writing a kinship handling, question-

answering program. Some simple kinship relationships may be expressed in predicate calculus as

Sibling(Jesper,Bodil)
Male(Jesper)
Father(Jesper) = Edvin
Wife (Edvin) = Edla

etc. ("Sibling" stands for "brother or sister"). For simple facts there is no essential difference in effort between making these predicate-calculus statements and making the obvious property-list storage instructions. However, in more complex expressions, like

Sibling( Father(Hedvig). Wife(Neighbor(Halvard)) )

(for "Hedvig's father is a sibling of Halvard's neighbor's wife) the predicate calculus formulation is probably more convenient than what we could immediately do with property-lists. This is what we mean in criterium (b) above.

Moreover, predicate calculus permits us to state general axioms, such as axioms that characterize these kinship functions and relations, e.g.

$(x)\ \ Male(x) \equiv \neg Female(x)$

$(x)\ \ Child(x,Father(x))$

$(x)(y)(z)\ \ Child(x,y) \wedge Child(x,z) \supset y = z \vee Sibling(y,z)$

If we write a direct program for the kinship exercise (without the support of any standard program), then the information contained in these axioms must somehow go into that program. Normally, it goes into the retrieval part, but some of the axioms could also go into the storage part. For a trivial example, the first axiom could correspond to a segment of the retrieval procedure which says "if it has been asked whether x is male, and if there is no immediate information in the data base saying that he is or isn't, then ask as a sub-question whether x is female, and negate the answer", and to a corresponding segment for the case where x is female. But alternatively, it could correspond to a section of the storage procedure which says "if you have to store that x is male, then store also that x is not female" and similarly for

the symmetric case.

If a small-data-base management system uses predicate calculus as its problem-oriented notation, then it should permit the user to write axioms like those above, and the system should be responsible for using the axioms in storage or retrieval. This is what we mean in criterium (d) above.

Let us now discuss some existing programs against the background of the aforementioned criteria.

The SPB program. This program has been developed within the SQAP project (Swedish Question-Answering Project) at FOA in Stockholm, and with our cooperation. The program is written in PL/360 (for IBM 360 series computers). It would seem possible, although not trivial to link the program with a LISP system. There is presently no sufficiently big, 360 compatible computer in Uppsala, so we can not take over the program directly.

For work in Uppsala, one possibility is to re-program some central parts of the SPB program in LISP or in the assembly language for some computer here, while retaining the same system design. In this system elementary facts must be quantified binary relations like

$$(\forall x \in a)(\forall y \in b)\ R(x,y)$$

This is a connection between the nodes (constants) a and b. Arbitrary quantifiers are permitted. For retrieval, the system can accept inference rules which are essentially implications ("if the following relations are present in the network, then the following ones can be added"). It also has a special facility for handling chaining rules. - This design works well for the purpose of the SQAP project, i.e. general-purpose natural-language question-answering, but several features would have to be added if the system is to be used as the basis for an assortment of specialized applications. We would need at least the following:

(a) a garbage-collection facility

(b) a possibility to store arbitrary expression (strings, list structures, or numbers) on the property-lists of nodes in the system, and standard functions for manipulating those expressions.

(c) substitute or add a new inference-rule feature, which is easier to use and more general than the existing one. In particular, it should accomodate the new types of expressions introduced in step (b).

The first two amendments come for free if we re-write SPB in LISP or some other existing language, but they would be cumbersome if we write in assembly language. (Actually, they would amount to building a programming language inside SPB, which is absurd). The third amendment is less trivial to handle. We could do it by kludges, but we should not, since we want to live with the system for some years. If we want to extend SPB in a general and systematic way, it seems better to go the other way: design a more general system, and see to it that the SPB notation can be defined as a subset.

The QA3 program. This program was developed at Stanford Research Institute. It is written in LISP, and we have a copy of it and could use it. QA3 is essentially a resolution theorem-proving program with clause indexing features which help the resolution program to find relevant clauses. QA3 permits the user to write facts and rules (formulated as axioms) in first-order predicate calculus, which is a more convenient notation than the one used by the SPB program. On the other hand, both test-runs and our analysis of the program indicate that retrieval using QA3 takes several orders of magnitude more time than if the same task and the same axioms are explicitly programmed in LISP. This means that QA3 can only be used for those applications which are absolutely impossible to handle by conventional methods. But this is unnatural, because most applications projects start with simple problems (for which the programmer is strongly tempted to write his things in LISP), and gradually grow into complex tasks. When the program reaches the limit of what can be done by conventional programming, it is probably too late to switch to the predicate calculus representation that QA3 requires.

What has been said here about the SPB and QA3 programs should not be taken as criticism. Those programs were developed for other purposes

and serve them well, so they fill other requirements than ours. - We have looked into a few other programs that might be useful, and made similar conclusions about them.

For these reasons, we have worked on the design of a system which would meet the four requirements on page E1. This work started in November, 1970. We now (February, 1971) have a general outline of the system, and manuscripts for parts of the program.

The proposed program is based on the following assumptions:

(1) It should manipulate formulas in first-order predicate calculus. We consider this to be a very convenient notation. The SPB data structure is a special case of predicate calculus, which is good for compatibility with our previous work.

(2) The program should be written in LISP, and operate on LISP's standard data structures. In principle, we could modify the LISP system and add special-purpose data structures for a "heavy" program such as this, but we will probably have to move around between several computers during the next few years, and the extra programming costs for going beyond the standard system would be too high.

(3) In order for the system to be attractive right from the start of applications projects, it shall contain facilities which handle trivial storage and retrieval operations, and handle them (roughly) as efficiently as manually coded programs. For example, if a user programs the kinship problem above in LISP, he would probably indicate the sex of a person by a flag (MALE or FEMALE) on the property-list of the atom for this person. Therefore, our program shall represent a predicate-calculus formula like "Male(Jesper)" in exactly that way, and it should do retrieval by immediately selecting an ordinary get operation, so that the user does not lose anything if he uses our program. (He probably does not win anything either to start with, but he knows that he works in a more powerful system, and therefore has a higher ceiling).

(4) The system should not be restricted to trivial cases in predicate calculus, but it should contain the full power of a resolution theorem-proving program. The general-purpose parts of the system

must of course be carefully integrated with the short-cuts which are needed to handle simple cases.

Since the p· pose of the system is to manage a small data base of predicate-calculus formulas, we decided to call it PCDB, for "Predicate Calculus Data Base".

Outline of PCDB. PCDB is a LISP program which maintains a data base of formulas in first-order predicate calculus, and which does retrieval in this data base by various means, including resolution but not excluding others. When compared to other "theorem-proving" programs, it stresses efficiency in handling relatively shallow deductions from a relatively large (hundreds, thousands) number of axioms. It does not stress completeness.

One characteristic trait in the system is that ground unit clauses ("facts") and other clauses ("rules") are stored in different ways. Facts are stored on the property-lists of their arguments. Thus the relation

Sibling(Jesper,Bodil)

will be represented on the property-lists of the atoms JESPER and BODIL. When handling the fact

Sibling( Father(Hedvig), Bodil )

the system will create a unique representation for the expression "Father(Hedvig)". This unique representation contains a property-list, on which the relationship can be stored. Rules, on the other hand, are assigned a name (which should be an atom), and stored on the property-list of this name. For example, if CHILFATH is the name of the rule

(x) Child(x,Father(x))

then this axiom would be stored on the property-list of the atom CHILFATH. Moreover, there would be references between the property-lists of the atoms CHILD, FATHER, and CHILFATH, which enables the system to retrieve this rule whenever it is needed.

A second trait in the system is to assign tailor-made function defini-

tions to predicate-calculus relations and functions. The definitions are tailored according to declarations which can be provided by the user, and which otherwise are given default values. For example, if the user wants the relation "Sibling", he should tell the system that this is a relation with two arguments, and he might add that it is not functional in either argument, that it is symmetric, and so on. The system then generates (at least) two function definitions that are somehow associated with the atom SIBLING: one storage definition, which is used whenever we want to assert a fact where the leading relation is "Sibling", and one elementary retrieval definition, which is used when we want to ask a question where the leading relation is "Sibling", and which simply checks whether the corresponding fact has been explicitly stored. Both definitions have been generated by the system, and are geared to conventions about where and how the "Sibling" relationship is to be stored; those conventions are selected according to the declarations.

A third feature which we would like to have, but do not yet know if we can provide, is compilation of rules. This would be desirable during search-for-retrieval. Such search can be performed by one general program which essentially takes the question; uses pointers from its leading relation and constituent functions (functions in the sense of predicate calculus); and selects rules which it attempts to apply (resolution-wise, these rules are axioms against which the question is resolved). However, it would be desirable to associate a third function definition with the relation and functions, namely a retrieval-search definition, which contains "compiled" versions of axioms which would otherwise have to be "interpreted". On the other hand, several difficulties arise if one tries to do this, and it is not yet clear whether such compilation is practically feasible.

So much about the PCDB project in general. Some comments about the schedule:

Present status. (January 31, 1971) Data representation and other crucial design decisions have been made. Parts of the program, including the piece that generates and optimizes function definitions for storage and elementary retrieval in relations, have been written but not debugged. A

very preliminary user's manual and program documentation for those parts of the system have been written and mimeographed, so that potential users of the system can give their comments as early as possible.

Modules and schedules. We have defined a number of relatively independent function packages, which are described here with their predicted completion dates. Dates refer to a reasonably debugged but not fool-proof system, and assume that two research assistants (försteamanuens) work at the project under the guidance of the undersigned (Erik Sandewall). All dates refer to 1971.

Network package: March 31
Generate function definitions for storage and elementary retrieval of relations.

H-expression package: March 31
Generate function definitions which handle what is functions in the predicate-calculus sense.

Clause storage package: April 30
Store what we have here called "rules", i.e. non-ground or non-unit clauses

Resolution operator package: May 31
Implement the resolution operator for combinations of "fact" and "rule" type axioms, with due respect to the tailor-made representations of "facts",

Retrieval-search package: First version May 31
Direct the search for an answer to a closed question. Tailor function definitions for retrieval-search from user declarations about properties (symmetry, transitivity, etc.) of relations.

Clause compilation package: November 30
Compilation of clauses as described above.

Function reduction package: First version March 31
Optimize the raw function definitions that have been generated by other modules.

Input/output package: First version November 30
Provide the user with a convenient infix notation instead of the standard LISP prefix.

The retrieval-search and function-reduction packages will have to be improved continuously.

Summary of schedule: A preliminary package which can do storage and retrieval, including some search, should be available by May 31, 1971. Continued work on the program will take most of the year 1971/72. A nice and reasonably complete program should be available and documented by June 30, 1972.

## F. Natural-language processing

We have formulated the following two goals for work on natural-language processing at DIU.

(1) Develop models for natural-language discourse;

(2) Develop tools which facilitate writing translators for special-purpose subsets of natural language in applications projects.

Let us discuss both goals in succession.

(1) Models for natural-language discourse. This goal was formulated as we participated in work on the SQAP question-answering program. That program (like other QA programs) assumes all input sentences to be of two kinds: assertions and questions. Assertions are statements which add something to the system's data base; questions are requests for fact retrieval from the data base. When several sentences appear in succession, the order is important for the interpretation of pronouns and pronominal constructions ("he", "the newcomer", "for this reason", etc.), but apart from this the order of sentences is irrelevant.

We have slowly realized (1) the belief that everything is either assertions or questions in this sense is a model of natural language discourse, and (2) that as a model, it is insufficient. This became painfully apparent as we took some sample natural-language texts and attempted to put them into the SQAP system. We realized that many sentences which have the form of assertions serve to emphasize something which the reader or listener knew before, rather than to tell him something new. In 'pure' assertions, there are a lot of interesting relationships between successive sentences. In a conversation, one person often makes statements in order to check that he has resolved some potential ambiguities correctly, and so on. These phenomena are characteristic of natural-language communication between man and man. Some model of them, however superficial, is very essential if we hope to provide a relaxed conversation between man and machine.

To make this more concrete, let us outline two models. One of them is very simple and can be realized with only a little programming, once a tool like the PCDB package of last section is available. The other model is relatively sophisticated, and it is close to the limit of what we can hope to provide.

The simple model attempts to describe how a person selects an answer to a closed question (answerable by Yes or No) in a simple conversation. This model is used in the SQAP program. For the purpose of the model, we assume that the listener contains a data base $\Gamma$ of facts, and a deduction procedure which checks whether a given proposition can be proved from the data base. If the proposition $\gamma$ can be proved from $\Gamma$, we write $\Gamma \vdash \gamma$. The deduction procedure gives either of three responses:

| | |
|---|---|
| $\Gamma \vdash \gamma$ | ($\gamma$ can be proved from $\Gamma$) |
| $\Gamma \nvdash \gamma$ | ($\gamma$ can not be proved from $\Gamma$) |
| - | (the procedure has not been able to determine what the case is) |

Suppose now that a question "is it true that $\gamma$ ?" is given to the listener. Our very simple model says that the listener then calls the deduction procedure twice. First it is asked whether $\Gamma \vdash \gamma$, and second it is asked whether $\Gamma \vdash \neg\gamma$, i.e. whether the negation of $\gamma$ follows from the listener's data base. After this, the answer to the question is determined by the following table:

| | $\Gamma \vdash \gamma$ | $\Gamma \nvdash \gamma$ | - |
|---|---|---|---|
| $\Gamma \vdash \neg\gamma$ | I'm confused | No | Don't know |
| $\Gamma \nvdash \neg\gamma$ | Yes | Don't know | Don't know |
| - | Don't know | Don't know | Don't know |

This simple model obviously does not give an adequate description of how humans <u>really</u> answer closed questions. It does however provide a crude

approximation. If a computer is programmed according to this model, then a human will think of its responses as "reasonable" and "correct", although certainly also "naïve" and "stereotype". The model gives us a basis for the program, and then we can think of how the model can be improved.

So much for the easy model. The difficult model is intended to give a partial explanation of a text like:

> "Be careful with the XYZ company. It is true that they are making record-high profits for the third year in a row. It is also true that their model ZYZ belly-buttons have been a roaring success. But the company is young, and they must market a good new product every few months to survive. ..."

Problem: what is the purpose of the "It is true that... " sentences? Let us first discuss what might be going on in the listener's mind, and then formulate the model. The input to the listener consists essentially of the following statements:

| | |
|---|---|
| $\alpha$ | You should be careful with the XYZ company |
| $\beta$ | They are making record-high profits... |
| $\gamma$ | Their belly-buttons are a success |
| $\delta$ | The company is young |
| $\varepsilon$ | They must market new products |

Of these statements, $\alpha$ is presumably the only real assertion, in the sense of something that was intended to augment the listener's data base. The other statements may well be known to the listener before, and they serve as support for $\alpha$, so that it shall slip easier into the listener's mind.

Let us assume, then, that the listener contains an acceptance procedure which checks every new "real" assertion and decides whether it should be let into the data base or not. The acceptance test is similar to the answering procedure in the first model, i.e. it tries to prove the assertion or its negation. To handle the present example, we must assume

that "proof" and "disproof" are performed, not in strict logic, but in a vaguer logic of "tends to support" (written as $\sim\!\!\rightarrow$) and the negated "tends to reject". In the above example, if the listener only hears $\alpha$, and if $\beta$ through $\varepsilon$ are in his data base, then his acceptance test will perform

$$\beta \sim\!\!\rightarrow \neg\alpha$$
$$\gamma \sim\!\!\rightarrow \neg\alpha$$
$$\delta \sim\!\!\rightarrow \alpha$$
$$\varepsilon \sim\!\!\rightarrow \alpha$$

and it will then have to balance these indications, presumably by assigning "weights" to them and combining them in a polynomial. Our model is now ready to explain why $\delta$ and $\varepsilon$ are in the text above: The listener might not know of them, or they might be stored in such a "passive" way in his data base that the "deduction" procedure does not notice that $\delta \sim\!\!\rightarrow \alpha$ or that $\varepsilon \sim\!\!\rightarrow \alpha$.

Furthermore, our model can crudely explain what the $\beta$ and $\gamma$ statements are in the text for. If they are not there, the listener might react like: "Oh, he claims $\alpha$, and he has $\delta$ and $\varepsilon$ for support. But he obviously does not know that $\beta$ and $\gamma$ . He is probably wrong". When $\beta$ and $\gamma$ are in the text, they block such a reaction. Their implicit frame is: "I, the speaker, know that $\beta$ and $\gamma$. I still claim that $\alpha$ ". The listeners' reaction could then be "Oh, he knows $\beta$, $\gamma$, $\delta$, and $\varepsilon$, and in his judgement $\delta$ and $\varepsilon$ are " heavier" indications than $\beta$ and $\gamma$ ." If the coefficients in the listener's "judgement polynomial" have been assigned low "reliability", then the listener is likely to conform to the speaker's judgement. But obviously the last word is still with the listener.

Formally, then, our model assumes (1) that whenever the "deduction" procedure derives an $\sim\!\!\rightarrow$ connection, it should also return indications of "weight" and "reliability" of this connection. The model also assumes (2) that the acceptance test contains a polynomial which balances these indications against each other and against the speaker's estimates. Finally, our model assumes (3) that a statement in the listener's data base which tends to reject the given assertion is given considerably

less weight if the speaker has mentioned that statement in an "It is true that" phrase.

The two models that we have discussed so far have been models of the listener's behavior. A generative model which describes how the sentence is created, must also contain a model of the speaker. Such a model necessarily assumes that the speaker has an urge to transmit a certain message and have it accepted. It should assume, also, that the speaker contains some model of the listener (which need not be identical to our model of the listener), and that he is able to use this model.

It can be claimed that models of this kind should belong to psychology, or linguistics, or philosophy, or any of a number of other disciplines. Moreover, representatives of each discipline can rightfully state that the sketchy models above are terribly crude, possibly even incorrect, and therefore of no value to his discipline. However, models like these have for us the advantage of being precise, operational, and therefore useful in our endeavour to write programs that carry on a decent conversation. A crude model is better than no model, in computer science like in any other science. This is why the construction of such models for natural-language discourse is one goal for our work on processing of natural language.

When we build such models for natural-language discourse, we want to test them on a computer. We therefore need a computer representation for the natural-language information, and we need an auxiliary program which takes care of storage and retrieval in the data base that pretends to be the "listener" in the model. The first criterium has been met by the PCF representation (see page B.6 of this report), and the second requirement is being met with the PCDB package (see section E). In both PCF and PCDB there is more work to do (axiomatization of additional primitives in PCF; addition of more features in PCDB), but we expect both "tools" to be usable by July 1, 1971. Therefore, this summer we plan to start using these tools for testing models of natural-language discourse, similar to the models that were outlined above.

(2) Our second goal is to develop tools which facilitate writing translators for special-purpose subsets of natural language in applications projects. Our applications typically require highly interactive programs, and they aim at manipulating data that do not have a trivial (e.g. numeric) format. It is therefore very desirable to provide nice, natural-language-like communication with these programs. On the other hand, we do not need to provide full natural language, since the domain of the program is very limited anyway. Experience shows that it is much easier to write a translator for a small subset of natural language, than for the whole language. What we want, then, is standard programs and other tools which facilitate writing such special-purpose translators.

One project along these lines is to set up and test Woods's Augumented Transitional Network Parser, which was described in the Communications of the ACM (October, 1970). We have a copy of the program, and Mats Cedwall is presently trying it on a test case. We expect that it can be included in the collection of useful tools. It should provide a framework for various sub-tasks in parsing.

Some other, and very obvious tools are the PCDB package (section E), the PCF notation (page B.6), and the various models for natural-language discourse that have been described in previous pages. A macro expander similar to the one used in Hans-Jürgen Holstein's ROBOT program is also desirable. It is available in FORTRAN and shall be re-programmed in LISP.

During the year 1971/72, we plan to work on a third set of tools, namely routines which handle some feature of some natural language in a cheap way. Examples of this are routines for handling "and" and "or" connections between nouns, or routines for handling the peculiar postfix definite article in Swedish ("en pojke" = "a boy"; "pojken" = "the boy"). In all such cases, we only expect our routines to handle the simple, regular cases. Irregularities and peculiarities are important for linguistic descriptions of a language, but they are a luxury in the input language for a computer program. - Tools in this set should work within the network parser or another, similar framework for the parsing process.

A fourth type of tool is also on the schedule for 1971/72, namely a

language learning program which takes new, previously unknown constructions plus an interpretation of them and which generalizes to an interpretation rule and stores away the rule in the data base, for use on later occurrences of the same construction. Such a tool would enable the user to extend the communication language gradually.

It is very difficult to estimate how much work will be required for these projects, and what the logical order of doing them will be. We shall therefore not set up any schedule for this work, but merely propose that work during 1971/72 shall concentrate on tool-building, particularly on tools for special language features and for language learning. People, ideas, and economical resources will determine how far we get.

## G. Formula manipulation

Work under this heading is dominated by the INTERFORM project, which is a combination of formula manipulation and data base handling. We shall now describe this project.

1. Introduction

2. INTERFORM

2.1 Short description

2.2 INTERFORM 1 (a subset of INTERFORM)

3. Time-table

3.1 Original time-table

3.2 Present status

3.3 Time schedule for 1971/72

4. The data base and search strategies

## 1. Introduction

In several applications, one uses mathematical models which are defined by a set of algebraic relations (e.g. equalities, inequalities) where it is not decided in priori which variable are given and which asked for.

For the study of such systems, one needs a programming system which provides algebraic manipulation and numerical evaluation of a general nature. However, programming languages like FORTRAN and ALGOL require an algorithm for the calculation of specified variables, when certain others have been assigned numerical values. One must choose such an algorithm before writing the program. Moreover, in such languages it is very complicated to express one variable symbolically in others.

A scientist, physicist a.s.o. could save lots of work if he had an interactive system which could perform the following tasks:

a) Save relations between variables in a data base.

b) Express some variables in others symbolically, using the relations in the data base.

c) Assign numerical values to variables.

d) Calculate variables numerically.

e) Choose relations which are relevant for a certain problem.

f) If the system doesn't have the information needed for solving a problem, it should ask the user for the missing information.

g) Calculate symbolic derivatives.

h) If possible, perform symbolic integration.

The goal for the research project is to build a system, which has the above described facilities. The system is called INTERFORM (INTERactive FORMula manipulation). Work at this system started during the spring of 1970 and is supposed to be finished during the autumn of 1972.

## 2. Short description of INTERFORM

The project is described in detail in "Artificiell Intelligens (1970)". In this chapter, an introduction to the project is given in order to describe the main ideas. To make the presentation more specific, we can think of a situation there we have a number of relations describing optical instruments, which we want to examine.

First of all, you give to the system a set of relations which describe a certain subject (e.g. optical instruments). The relations are usually equations where one variable is expressed as a function of others. Let us associate this relation set with a name (M1) and the relations with names (R1, R2, R3,....). Assume also that some variables have had numerical values assigned to them. Then you can give instructions such as:

| | |
|---|---|
| COMPUTE | All the relations needed for computing F numerically are taken from the set M1. If it is not possible to compute F numerically, the system asks for the values of the unknown variables on which F depends. |
| EXPRESS F IN A AND B | All the relations needed to express F in A, B and as few other as possible, are taken from the set M1. |

You can also define subsets of relations, combine sets, and create new sets. In fact the data base contains a set of sets of relations. One of these is called the work set and is understood if nothing else is said. Examples:

| | |
|---|---|
| COMPUTE F | Search in the work set for relevant relations. |
| EXPRESS F IN A USING M3 | Search in the set M3 for relevant relations. |

If the data base describes optical instruments, one set (M1) may describe field-glasses, another set (M3) may describe microscopes and so on.

Binding variables to values (numerical or symbolic) can be done temporarily or permanently. Examples:

a) COMPUTE F WHEN A=2, B=5

or

SOLVE F

GIVE A,B (question from the system to the user)

A=2, B=5

In these two examples, A and B are bound to 2 and 5 only during the evaluation of F.

b) BIND A TO 2, B TO 5

COMPUTE F

A and B are bound to 2 and 5. This holds until next BIND A .....

Suppose we wish to bind certain variables to certain values in order to describe one particular subject which the relation set describes. E.g., if M1 describes field-glasses, binding N to 3 and MY to 1.3 may describe field-glasses with three plastic lenses, and binding N to 2 and MY to 1.5 describes field-glasses with two glass lenses. It is now reasonable that you should be able to define sets of binding variables, and to use these sets temporarily or permanently. Examples:

COMPUTE F USING B1 — B1 is here a set of variables bindings. These variables are temporarily bound to the values respectively.

BIND B2 — B2 is a set of variable bindings. These bindings are made permanent.

Another task for the INTERFORM system is to answer questions like:

"On which variables does F depend?"

"Which variables depend on F?"

"Does F, G or H depend on A, B or C?"

In the last case, the answer is a matrix of the following type:

| | A | B | C |
|---|---|---|---|
| F | YES | NO | NO |
| G | NO | YES | NO |
| H | YES | YES | YES |

In the first case, you will get a print-out of the relation structure similar to:

```
F                                   F is defined in R3 and R4.
    R3                              In R3, F depends on A, B and C.
        A                           A and C are uddefined but B is
        B                           defined in R4 and R5, etc.
            R4
                Q1
                Q2
            R5
                Q3
        C
    R4
        H1
        H2
```

and in the second case:

```
F
    R6 : G=f(F)                     F occurs as a right hand variable in
         R7: R=f(G)                 R6 and R9. In R6, G is defined. G is
         R8: N=f(G)                 a right hand variable in R7 and R8
    R  : H=f(F)                     etc.
         R10: A=f(H)
         R11: S=f(H)
```

The search for the relation structure is mainly an INTERFORM 1 facility (see 2.2 below) where the distinction between right hand and left hand variables is relevant.

Before leaving this chapter, we shall indicate the main difficulties. These arise when we want to do "EXPRESS F IN ....." or "COMPUTE F FOR...", and when we want the system to find the most relevant relations for the problems. The two main problems are:

a) If we can express F in A, B, C in several ways, which is the "best" way to express F?

b) How should we handle situations where F depends on a variable, which depends on F?

These questions were part of the reasons for defining a subset of INTERFORM, namely INTERFORM 1. In chapter 4 we shall outline how INTERFORM 1 treates these problems.

## 2.2 INTERFORM 1

The following limitations are proposed for INTERFORM:

a) All definitions of relations should have the form:
<relation name> : <left hand variable> = <arithmetic expression>

b) Variables which never occur as left hand variables, are considered as undefined.

## 3. Time-table

### 3.1 The original time-table was

| | |
|---|---|
| spring 1970 | Reading. Discussions with prospective users to find out about desirable features. Dividing the project into modules (representation of the data base, formula manipulation modules, I/O modules etc.). Programming of simple modules for batch use on the CDC 3600. |
| autumn 1970 | Continued programming and test runs in batch on the CDC 3600. |
| 1971 | Programming the interactive modules (I/O-modules etc.). Test runs on the Siemens 305 (interactive LISP system). |
| spring 1972 | a large number of test runs. |
| during the whole period 70-72 | Linking of the modules in the system. Documentation. |

### 3.2 Present status

Extensive reading was done during the spring of 1970. We have been in contact with potential users but that has not given us any further proposals.

During the autumn of 1970 we have representation of the data base and defined the modules for INTERFORM 1 (see 2.2). Some of the modules are

ready and can be demonstrated on request. These are:

a) Input of the relations to the data base.
b) Relation analysis of the type "which variables is F depending on?" (page G.4)
c) The COMPUTE and EXPRESS commands explained above under the condition that every defined variable is defined only once.

Input and output is still in LISP notation (prefix and fully parenthesized).

There is no simplification of expressions (we intend to use existing simplification programs).

3.3 Time-table for 71/72

The following time-table assumes one person working full-time and one working half-time.

| | |
|---|---|
| spring 71 | a. Study of different heuristic methods for search in AND/OR-trees (which are needed for INTERFORM 1). |
| | b. Implementation and testruns of these. |
| | c. Implementation of a non-heuristic (exhaustive) search procedure. |
| | d. Time-studies of the programs developed in b and c. (Does the heuristic procedure itself take so much time that it is not worth while?) |
| autumn 71 | a. Implementation of I/O-programs, including the simplification package. |
| | b. Test-runs, documentation. |
| | c. Interform 1 finished. Test-runs by possible users. |
| spring 72 | Extension of INTERFORM 1 to INTERFORM (Don't use the restriction that a variable is undefined if it occurs only as a left-hand variable). |

autumn 72 A large number of test runs. Documentation.

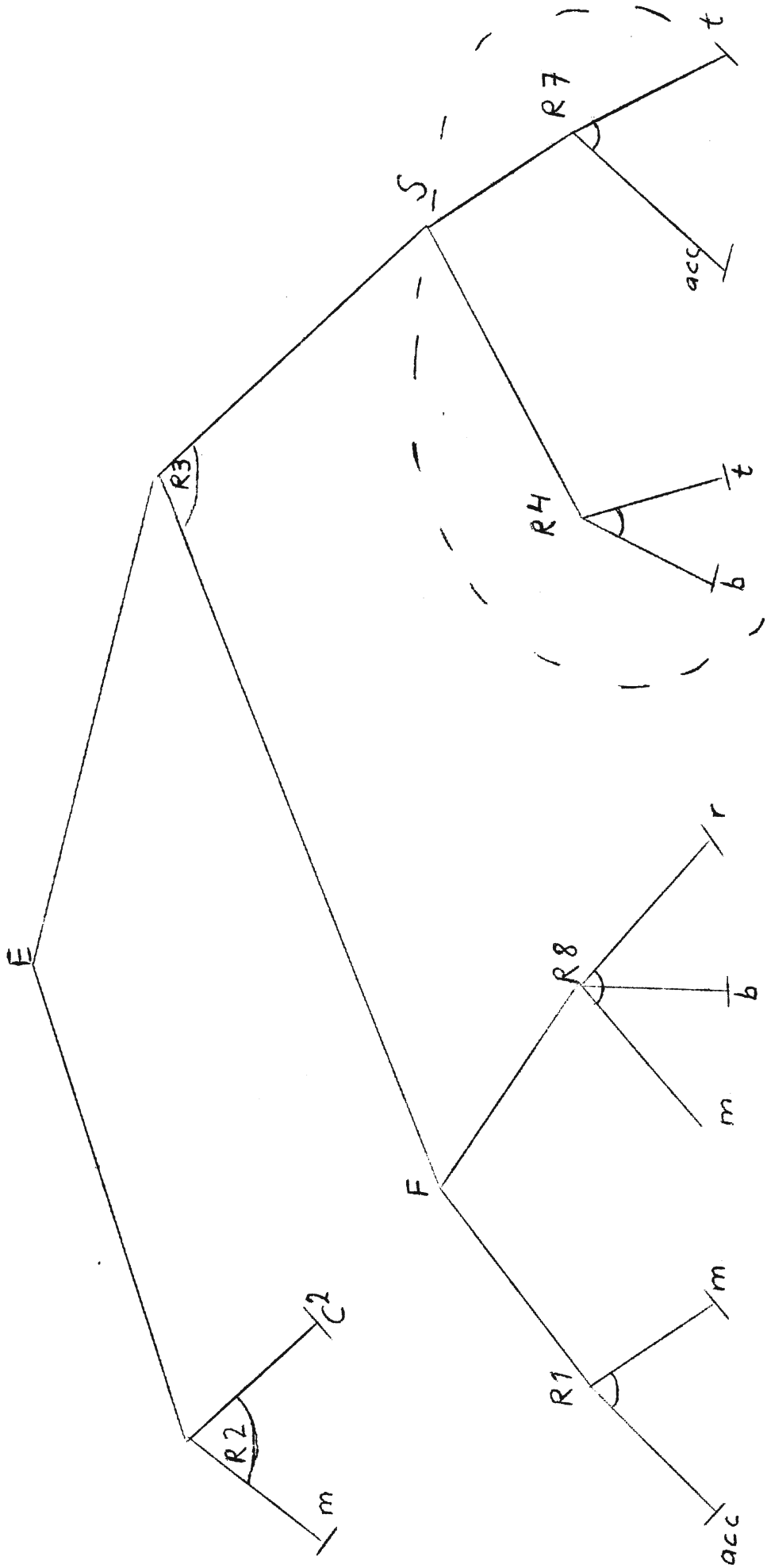
4. Outline of the data base and the search strategy.

Consider the following example of a relation set:

| Name | Equation |
|---|---|
| R1 | $F=acc \cdot_2 m$ |
| R2 | $E=m \cdot c$ |
| R3 | $E=F.S$ |
| R4 | $S=v \cdot t$ |
| R5 | $W=E/t$ |
| R6 | $v=acc \cdot t$ |
| R7 | $S=acc \cdot t^2/2$ |
| R8 | $F=m \cdot v^2/r$ |

To each relation name we assign a list of the corresponding right-hand variables. To each left-hand variables we assign a list of names of the relations where the variable has been defined.

The following tree-structure describes how E depends on other variables:


E
R3
S
R7
t
acc
R4
t
b
R8
r
b
m
F
R2
C2
m
R1
m
acc

E might be expressed using R2 or R3. Using R2, we have to express both m and c. That means that we can choose between relations, but (in principle) we have to express all right-hand variables in the choosen relation. This is represented by an AND/OR tree, where the AND nodes are marked with an arc. Of course it is now possible (and certainly the most common situation) that there are loops in the tree. It t is a left-hand variable and s is one of its right-hand variables, we have an example of such a loop. The idea is now to use methods for heuristic search in AND/OR-trees with common subnodes. Such methods exist [1] for the case where solutions to AND-connected branches are independent. We are now working to generalize the method to the case where solutions to AND-connected branched are mutually dependent.

---

(1) James R Slagle
Deena A Koniver } Finding Resolution Proofs and Using Dublicate Goals in AND/OR-Trees.

## H. Qualitative analysis of models

### (1) The Use of Sociocybernetic Research

As our society changes more and more rapidly and grows more and more complex, it also becomes increasingly important that our power-holding decision-makers and their experts and planners can be adequately informed. Increased societal complexity implies increased interdependence of societal subsystems. It is therefore very important that social conflicts can be avoided; this, in turn, requires that the consequences of societal reforms and development projects or policies are well analyzed before one sets out to realize them. However, the faster the societal change rate, and the more voluminous the stream of unstructured ("raw") information poured out by mass media, the less is the chance for central decision-makers to be 'rational', that is: to be adequately informed and to be able to judge all objectively given alternatives adequately. The price we pay for suboptimal society control is evidently social problems.

The only ray of hope for an improvement (instead of a continuing deterioration) of this situation comes today from sociocybernetic research. Spectacular results cannot be expected for the next couple of years, it is true. But a non-trivial solution in the form of an automatically data-analyzing and theory-synthesizing information system lies not further than twenty effective man-years ahead.

The Research Group on Sociocybernetics within the DLU has existed for two years now. It has been funded by the "Riksbank Tercentenary fund" as a part of a relatively large sociological project on empirical study of social mobility ("Märstaprojektet"); however, the Research Group has grown more and more independent as it became more and more evident that the Märsta Project's information-processing and modelling difficulties were universal not only among social researchers, but also among behavioral scientists, ecologists and environment planners, and even among managers and politicians.

The Sociocybernetics Group (RGSC) maintains research in three directions:

(a) methodological research, i.e., investigations of the logical and practical possibilities of automatization of social research;

(b) detailed modelling of "biopsychosocial" behavior of humans;

(c) multi-levelled modelling of social/ecological/economical processes ("macroprocesses").

But the RGSC has actually only a single goal: the automatic system which integrates the results of the above-mentioned three mutually complementary subprojects. There is a unique consistency of the Group's every single research effort with the integrated-system project. (Most of DLU's general projects can furthermore find obvious applications within the RGSC's project.)

## (2) The Subprojects

### (2.1) Methodology. Automatic Data Analysis

We expect sociocybernetics to complement (or even to innovate) 'orthodox' sociology quite significantly, with regard to both methodology and theory. Contemporary 'orthodox' sociology comprises itself a broad methodological and theoretical repertoire, which, however, is mastered by only few in its entirety and which therefore is exploited far less effectively than most social researchers would like to. Social researchers usually feel that they lack the time for exact orientation and for experimentation with various schemes for analysis; the data need to be analyzed while they are still 'hot' enough to be of any practical use and interest.

There is also a lack of easy-to-use computer programs for complex analyses of data; there exist, f.i., to our knowledge, no programs which automatically (that is: without explicit commands) generally test the assumptions and adequacy of different statistical schemes of analysis, then perform those analyses which the tests recommend, and finally after listing the results, try to combine them in a problem-oriented, not method-oriented, comment.

The RGSC has a program of this type as one of its research goals. It will find two uses in the final system: To relieve the researcher of the incredibly time-consuming but relatively unqualified data-processing drag in order to release his capacity for the interpretation of the analysis results; and to feed structured information on empiry into the system's data base (where it is accessible later on during the system's theorizing endeavors).

A kernel program of great user-oriented flexibility and maximal machine-design independence has been implemented. The kernel program administrates master-card interpretation, dynamic storage allocations, and data reading, and it gives basic computational capabilities. It does not, however, include programs describing specific statistical tests and techniques; these are routines to be monitored by, not included in, the kernel program. Any number of specific routines can be temporarily or permanently linked to it with great ease.(This kernel program is called ROBOT3; it is realized in the form of a task-description interpreter, written in a maximally CDC/IBM-interchangeable FORTRAN. However, there are plans to rewrite ROBOT3 in a special FORTRAN-based LISP for additional power and unlimited extendability.)

(2.2) Macromodelling with a Multi-levelled Simulating System

The RGSC is not primarily concerned with ordinary simulation modelling. A preferred objective, in line with the Group's explicit ultimate goal, is the implementation of a sociologically general multi-levelled simulating system of utter flexibility, extensibility, and ease to use.

To be more explicit: The RGSC works on the implementation of a simulating system(called ATMAN) with database-handling capabilities. In this system there is no essential difference between a statement and a simulated entity. The system shall be able to switch from making deductions by formal rules to making deductions by simulating the consequences of a given situation. The system shall also be able to switch between different levels of detail, and to improve its predictive reliability by sampling over alternative competitive part-models (in a fashion already employed to obtain reliable technical systems made up of unreliable components).

(2.2.1) The Plausibility-Estimation Feature

The designed simulating system ATMAN is intended to be able to answer questions, through formal deductions or through simulations. It shall also be able to estimate the plausibilities of its own answers or of submitted hypotheses or theories.

A first program which can rank hypotheses and theories according to their plausibilities has been implemented by the RGSC; it does demonstrate the feasability of such programs, but the range of theories it can handle in its present form is very limited. A much improved and quite general version is at present under design by the RGSC.

(2.2.2) General Applicability

There has been growing interest from the DLU-colleagues involved in the planning of the ecological project in the accelerated development of the ATMAN system. It was therefore decided that the projects should collaborate under 1971 for common gain.

(It is to be expected that, in the course of time, also other projects will become interested in such collaboration. We think especially of the use of an ATMAN system in medicine, where a given patient's data (physiological, psychological, social) can be entered and contrasted with different hypothetical treatments; ATMAN could, f.i., be used to test whether a certain set of medicines is (according to all knowledge) consistent with the physiology of the given patient; a plausibility-estimate will be ATMAN's answer.)

(2.3) Micromodelling

The micromodelling efforts within the RGSC are aimed at supplying the very contents (in the form of explicit and operational process and system definitions) of the ATMAN system's database. Those who have never tried can hardly imagine how redundant ordinary psychological and sociological literature is from this point of view. Social reality must be reconstructed in the form of a single, connected and consistent model, from the biological level up. It is indeed a formidable task. But the computer

puts contemporary social researchers into an essentially different position than philosophers, psychologists and sociologists have been in in past decades and centuries. Structured knowledge can be accumulated in the computer in the form of model modules, level of detail of the definition of one module can be changed without invoking the necessity of redefining other model modules. Macrophenomena need never be explicitly deduced - they are automatically generated in the interaction of micro-processes. And the computer does not put as narrow a limit on details to be considered as does the human mind.

During 1970 there has been a concentration upon the micromodelling of interpersonal perception, of coalition formation, and of motivation. However, none of the models is as yet sufficiently formalized to be acceptable by ATMAN. This remains to be done during 1971. Micromodelling resulted in 1969 in formalized models of social power and of interpersonal attraction, and in a tentative measure of democracy. The next areas which the RGSC will attempt to micromodel are social justice, socialization and resocialization, and concept formation (including attitude formation and rule learing).

## Conclusion

The problem field of sociocybernetics is vast, and it may seem over-ambitious even to try to cover it with a single, rather small, research project. In this respect the RGS is purposefully in opposition to the general trend towards more and more particularized research. It is today far more important that we arrive at a single coherent, but not necessarily consistent, and perhaps not yet even quite correct model of social reality rather than obtain additional models of details of social reality, which in a more genral framework may be quite trivial. Of course, the RGSC is only meaningful as a long-term project; hopefully it will manage to accumulate knowledge and to synthesize the long-desired operational common base for social research.