ABSTRACT

The aim of this study was to develop a model for the mathematical word problem-solving from logical point of view. Word problems and the underlying equations were shown to be isomorphic.

We studied the relevant theories related to word problem-solving model. First a survey of the history of word problems and corresponding computer programs was conducted. The properties of logic programming as a modelling language was then investigated. The model was developed on the basis of that theoretical groundwork.

The model is relational, and has several abstraction levels. The model is called the TEACHER, because we want to emphasise the similarity of the model and Polya's strategy to teach problem-solving. Questions concerning natural language understanding are not covered. The TEACHER is able to generate exactly those word problems that it is can analyse. Text analysis and text generation of the word problem is done with logic grammars.

The result of this study, a logic based model of mathematical word problem-solving, could be used in different ways. First, the model is a confirmation of Polya's way to teach problem-solving. Second, the model may be used as a starting point for further theoretical development and as a framework for empirical studies of word problem-solving. Third, the TEACHER is a computer program that can be used a basis of an Intelligent Tutoring System.

CONTENTS

1. INTRODUCTION 1

1.1 A logic based model of mathematical word problems 1
1.2 Introduction of the subject and its importance 3
1.3 Short introduction of previous results 4
1.4 Approach method and research objectives 6
1.5 Results 8
1.6 Division to chapters 10

2. RELATED WORK — HISTORY AND MOTIVATION 12

2.1. History of word problems 12
2.2. Computer Assisted Instruction 15
2.3. Situation Problem Solver 21
2.4 A prognostic view of word problems 25

3. PROLOG AS A MODELLING TOOL 31

3.1. Pure Prolog as a modelling tool 31
3.2. Extensions of Pure Prolog 36

4. THE CONSTRUCTION OF THE MODEL 43

4.1. The language of word problems and the problem domain 44
4.2. DCG-grammars 47
4.3 A mapping between an equation and a word problem 54

5. THE BI-DIRECTIONAL SYSTEM 59

5.1 The TEACHER as a word problem generator 59

5.2 The TEACHER as a model of word problem-solving processes 61
5.3 The TEACHER as a test bench of problem-solving processes 67

6. SUMMARY 70

REFERENCES

1. INTRODUCTION

"You can only find truth with logic if you already found truth without it."
G. K. Chesterton

1.1. A LOGIC BASED MODEL OF MATHEMATICAL WORD PROBLEMS

This work deals with a fundamental problem in school mathematics: *How to transform a mathematical word problem into an equation*? [HaKi-89, KiGr-85, NaKi-92, Poly-65, Reus-88, Thae-86]. Our aim is to build a logic based model between an equation and a word problem. There are two didactic reasons that support the choice of word problems. First, word problems are much more difficult for students than comparable algebraic problems [HaKi-89, Reus-90]. Second, the significance of word problems is rapidly growing in secondary and high school curricula [Anon-93]. Thus the main emphasis is on the pragmatic needs of the modelling of problem-solving process rather than on general cognitive theories. However, Cognitive Scientists as well as researchers in Artificial Intelligence and Computer Science have a rich tradition in the investigations of mathematical word problems [AnBo-90, BaFe-81, Bobr-68, BrLa-84, CoVe-86, MiCa-83, PsMa-88, Reus-87b, Simo-83, Scho-87]. We use this knowledge as background information to build a theoretical model that has some practical implications.

This work is interdisciplinary. We use the methods of Computer Science and apply them to Educational Science. To be precise, we use the general methods of

Artificial Intelligence (AI) and especially the art of Logic Programming [Brat-86, ClMe-81, GeNi-88, Kowa-79, StSh-86]. The common denominator of AI and the Didactic of Mathematics is problem-solving. We shall model the problem-solving process of mathematical word problems that are presented in a textual form.

The starting point of our analysis is the structure of an equation. The equation is an equivalence relation between two objects that have an inner structure with sub-objects, relations between sub-objects and operations that bind these sub-objects together. The identical structure can be found from the word problem. In other words, a word problem is isomorphic with the corresponding equation. Therefore it is possible to find the logical structure of a word problem.

The starting point of the previous models of word problem-solving has been the textual form of an equation [Bobr-68, BrLa-84, CoVe-85a, HaKi-89, KiGr-85, NaKi-92, Reus-87b]. We are analysing the logical structure of a word problem and the underlying equation simultaneously. This enables us to build a relational, or bi-directional, model. A model is called *bi-directional* if it has two different interpretations: from a word problem to an equation, and from an equation to a word problem. We are using logic programming as a modelling language. As a result we get a model for analysing the mathematical problem solving from logical point of view. At the same time we also get a computer program that can build an equation from a word problem. Because the model is bi-directional then so is the computer program. Thus the program also transforms a given equation to all possible word problems that it can analyse.

We are not trying to solve questions concerning natural language understanding. The word problems presented in a textual form are, however, required to be analysed with a computer program. To do this, we have to create an artificial language of word problems. It is possible to define the artificial language using Definite Clause Grammars [PeWa-80, ClMe-81, McCo-87, Brat-86]. At the same time we get a computer program that can generate all the possible sentences of the artificial language. We call our prototype the TEACHER[1]. It can analyse only a very restricted sub-set of natural language. The TEACHER can analyse only those word problems that it can generate.

We are not modelling a human problem-solving processes with a computer. The model can be used and understood without a computer. However, it is not easy to

---

[1]We also call the corresponding model with the same term TEACHER.

separate the computer program and the model[2], because we are building a model using the methods of logic programming. The computer program may be seen as a validation of the logical correctness of the model. In addition, the computer program has some interesting features such as the ability to generate hundreds of meaningful word problems from one equation.

1.2. INTRODUCTION OF THE SUBJECT AND ITS IMPORTANCE

Much importance has been placed on mathematics education and on mathematical word problems in the last few years [HaKi-89, Lehn-88, Mali-92, NaKi-92, Pea-87, Scho-87, SeBe-91, Silv-87, StSl-91]. The curricula are changing in Finland and in the International Baccalaureate (IB) high schools around the world during the next few years (1994-1995) [Anon-93]. The curricula in mathematics are changing partly because of the use of computers. Many of those tasks that can be better performed with computers, such as curve-drawing and basic manipulation of polynomials, are losing their importance. Problems that are presented in textual form, mathematical word problems, are thus becoming more and more important in mathematics education at all levels.

Subjects where man is superior to machine, like natural language understanding and problem-solving, have been the ambitious research topics of AI from its early days [BaFe-81, Chom-57, Colm-78, Elor-74, NeSi-63]. Word problems have been studied by the researchers of AI, e.g. [Simo-83, Bobr-68], basically for the same reason as these problems are now gaining importance in education: namely, "solving problems is the specific achievement of intelligence and intelligence is the specific gift of man" [Poly-65].

It has been shown that many students solve algebraic equations without understanding why the different procedural details are justified [StSl-91]. It is not difficult to master the procedural way of solving an equation. The difficulty is in the conceptual understanding of equivalent equations [StSl-91]. This is the main reason why our starting point is the equation seen as an equivalence relation between two objects. Students who already master algebraic formalism also have

---

[2]We use the phrase the TEACHER system to denote both the TEACHER prototype and the TEACHER model.

difficulty to find the mathematical entities and relationships presented or implied in the problem text [HaKi-89].

Students, who are taught to solve word problems considering mathematical relationships between pairs of quantities, solve certain kind of word problems efficiently [SeBe-91]. We want to emphasise the role of relations in our model, because the understanding of relations seems to be the key factor for a true understanding of the problem-solving process.

1.3. SHORT INTRODUCTION OF PREVIOUS RESULTS

Computer scientists, especially those who have been specialising in AI, have been interested in word problems for various reasons. In the 1960's and in the 1970's the main interest areas were computational linguistics, e.g. pattern matching [Shap-87, p. 718-719] and story analysis [Shap-87, p. 1090-1099], and problem-solving [Elor-74]. Bobrow's STUDENT was able to solve high school level word problems [Bobr-68]. Bobrow's aim was to discover how to build a computer program that could communicate with people in a natural language within some problem domain. Charniak developed Bobrows work further and "foreshadowed an interest on a part of the AI community in the 1970's in story understanding" [Shap-87, p.135].

From the mid 1970's mathematical, physical and chemical school-book problems have been studied in the field of machine learning [MiCa-83, MiCa-86]. The problems of natural language are kept aside, although many researchers have developed a natural language front end [MiCa-86]. Thus the word problems, in a strict sense, are not included in these studies. However, these studies may enlarge our understanding about human learning and problem-solving [MiBa-89]. These issues are essential in machine learning:

> *I would give a very high priority to research aimed at simulating, and thereby understanding, human learning. It may be objected that such research is not AI but cognitive psychology or cognitive science or something else. I don't really care what it is called; it is of the greatest importance that we deepen our understanding of human learning, and AI community possesses a large share of the talent that can advance us toward this goal* [Simo-83].

Educational scientists have recently found the computer metaphor as a valuable tool to model, and thus to better understand, the basic questions of learning [Duff-90, Egge-90, Mali-92, NaKi-92]. Reusser implemented in the mid 1980's the Situation-Problem-Solver (SPS), a computer simulation model of understanding and solving word arithmetical problems [Reus-87a, Reus-87b, Reus-88]. Reusser's studies in word problems were part of the broader research project "on theoretical relations between text comprehension, situation comprehension, and mathematization in solving word arithmetic problems, and on the teachability of these issues" [Reus-87b]. Reusser has continued his work on the psychological processes of how students understand and solve mathematical word problems [Reus-88]. He has also developed an Intelligent Tutoring System (ITS) that uses the ideas of the SPS-model [ReKä-90].

The SPS was one of the first attempts to combine the methods used in the field of AI with the research on mathematics education. The model had, however, some weaknesses (Section 2.3). We want to build a parallel model to the SPS in which *we are not considering the complicated time dependent reasoning*. The SPS can analyse only a fixed number of predefined problems. We want to broaden our model in this issue. Even Bobrow [Bobr-68] saw that word problems found in standard textbooks form a semantically poor subset of natural language. Thus, it is possible to define the language of mathematical word problems, or more formally: *to write the grammar of the language that our model is able to analyse*. The third difference between the SPS and our model is the most fundamental one: the SPS analyses the word problems and the result of this analysis is an equation. We argue that the problem-solving process is not uni-directional. The problem-solver has sometimes to go back and re-evaluate the previous facts. Therefore we want to build *a model that also looks at the problem in another direction, from the equation to the word problems* (Fig. 1). We are building a model that can handle these kinds of relational features (Chapter 4).

1.4. APPROACH METHOD AND RESEARCH OBJECTIVES

Lewis [Lewi-81] used the word "algebra problem-solving", when he studied man's ability to solve algebraic equations. In this study we are mainly interested how word problems are "translated" into equations and not how these equations are solved. However, the thoroughly examined area of solving equation can be seen as valuable background information [Matz-82, MiPa-89, AnBo-90, PaSq-90].

How to solve mathematical word problems? This question has been actual for hundreds of years. Our starting points are the traditional methods for solving mathematical word problems, nicely combined and further developed in Polya's books [Poly-45, Poly-65], and the general theory of problem-solving and heuristics (with many examples from mathematical word problems) [Elor-74].

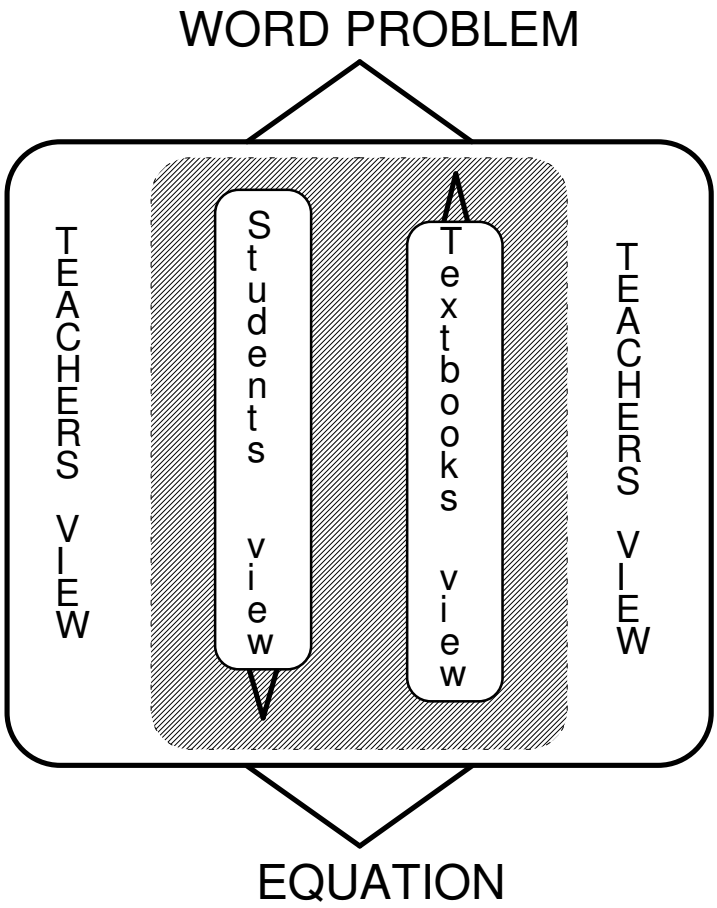

Figure 1. The Teachers view to word problems is a combination of the students pursuit to solve the problem and of the textbook writers intentions to find word problems that the student is able to solve.

Our objective is to better understand the problem-solving process that is needed to transform a mathematical word problem into an equation. Our perspective is that of a teacher. Therefore we first have to analyse ordinary textbook problems, then build a step by step model *from text to equation*. Or rather, from the point of view of a teacher, we know what kind of an equation a student is able to solve at each level. We have to find a corresponding word problem, whose partial answer is that equation. In other words, we are analysing the process step by step *from equation to word problem*. But a real teacher is not allowed to forget the view of the student (Fig. 1). To understand how word problems are solved we need to combine the view of a textbook writer (equation → word-problem) with the view of a student (word problem → equation).

Our main objective is

to build a bi-directional model between a word problem and an equation.

The model will be logic-based and we need tools for analysing and parsing the text. The previous models of word problem-solving processes, e.g. the STUDENT and the SPS, have originally been implemented by LISP, a functional programming language. LISP has been used by linguistics over two decades. LISP is also a language that has been widely used by researchers in AI [BaFe-81]. The long and common tradition of the LISP, linguistics and AI have also had an impact on the computational models. The main feature that can be found in these models is the use of functions. However, many of the present models of word problem-solving processes have relational features [HaKi-89]. Relationality can most naturally be handled by logic programming [StSh-86]. In this work we use the logic programming language PROLOG (PRogramming in LOGic) to build a transparent, relational (bi-directional), program.

> *Logic programs can be made transparent. In other words logic programs use facts to represent the static situations and rules to represent the inference processes between these situations. Logic programming uses a language that can be understood by people that are not familiar with ordinary (functional) programming languages* [Kowa-79].

The starting point of traditional word problem solvers has been the textual form of word problem. Therefore, the history of mathematical word problem solvers is intertwined with the basic problems of natural language understanding. Our

objective ***is not*** *to try to understand and answer questions concerning natural language text comprehension*. However, we do not only want to consider fixed problems (like SPS). Our aim is to create concise grammars that may be enlarged to handle a semantically poor subset of natural language, the mathematical word problems found in standard textbooks [Bobr-68].

Grammars for natural and artificial languages can be written as logic programs. Grammars that may be analysed by Prolog are called *logic grammars* [McCo-87]. The basic version of logic grammars, Definite Clause Grammars, briefly DCG, is a grammar formalism and at the same time effectively executable in Prolog. The most Prolog implementations offer a build-in DCG-version. We use this tool for text analysis and text generation. We want to emphasise that our model knows all the problems that it will encounter. Problems that do not follow the grammar of the system are out of the systems boundaries.

Our objective is to build a simplified test environment for modelling the word problem-solving processes. In doing this we have chosen the part of the curriculum [Anon-82] where mathematical word problems are to be solved by first forming an equation. This is another difference between the previous models and our system. The previous models have mainly been interested in elementary arithmetical word problems [BrLa-84, CoVe-85a-b, Reuss-87a]. We concentrate on the first introduction of algebraic word problems. This part of the curriculum is important for further mathematical studies. Many high-school algebra errors of solving equations are due to the conceptual changes involved in moving from arithmetic to algebra [Matz-82].

1.5. RESULTS

We shall build a simplified test environment, called TEACHER, between a word problem and an equation (Chapter 4.). The TEACHER system has been designed and implemented by logic programming. The main result of this study is that it is possible to model a problem-solving process and simultaneously get an executable Prolog program of that model. In fact, it is impossible to separate the abstract modelling process (TEACHER model) and the implementation process (TEACHER prototype) with Prolog. This is due to the features of the model. First, the problem-solving process proceeds through abstraction levels (Fig 2.)

Second, deductive reasoning is needed within and between these levels. Third, the model is basically relational. All the above mentioned three features are embedded in the art of logic programming. Conversely, logic programming is a useful tool in refining the model, because the process of implementation forces one to make an in-depth analysis of the model.

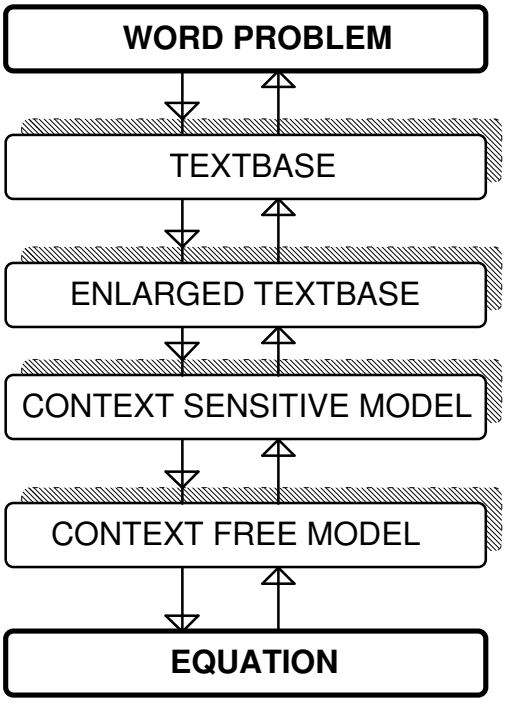

Figure 2. The abstraction levels in the TEACHER model.

The TEACHER transforms a word problem bi-directionally from text to equation using the same steps as a teacher introduces to a student. The bi-generative architecture of the TEACHER system (Chapter 5) can be viewed in many different ways.

First, it may be seen as *a test bench of problem-solving processes in a restricted area*: Is it possible to know when extra knowledge is needed and when extra knowledge is not needed to solve the problem? What are the steps that are related to the choice of the variable and how many steps are there? How is the variable selected? What are the psychological and technical, or contextual and textual, details that have an effect on the choice of the variable?

Second, the TEACHER system may be seen as *a model of the mathematical word problem-solving process with phases that a teacher introduces to his students*. These phases follow quite closely to the well-known Descartes' scheme [Poly-65]. The transparency of the Prolog-code makes the features of the model explicit. Thus the TEACHER forms a basis for an Intelligent Tutoring System (ITS).

Third, the system may be used as *a word problem generator*. This is due to the bi-directionality of the system. From a given equation the TEACHER prototype generates a group of equations all of which might be found in standard textbooks. The group of word problems that the TEACHER prototype is able to solve, is explicitly defined by a logic grammar formalism i.e. one can say in advance, by studying the logic grammar, whether a certain problem is solvable by the TEACHER. It is easy to enlarge the logic grammars to handle problems of different languages, in this version Finnish and English, or to widen the scope of problems the TEACHER prototype is able to solve.

It should be noted that the three features are not separate. For example, the generation of equations may also be used as a module in the ITS part of the TEACHER. It is also possible to use the TEACHER as a test-environment for the studies of problem-solving processes. This can be realised by keeping the underlying mathematical structure invariable and varying only the semantics of the text. The TEACHER prototype at present is just a framework for building a larger environment. The motivation of this study is in forming a new type of a model for doing all this.

1.6. DIVISION TO CHAPTERS

In Chapter 5 we analyse and discuss the above mentioned three features of the TEACHER system in detail. Then the restrictions of the model and its implementational issues are discussed. The special emphasis is put on the common factors of Polya's way of teaching problem-solving and the TEACHER as a logic based model.

In Chapter 2 we take a closer look into the history of previous works that have combined computer programs and mathematical word problems. The history of Computer Aided Instruction is then covered. The Situation Problem-Solver, an

inspiring example for the present work, is discussed in more detail. Finally, the impact of computers and graphical calculators to modern curricula is surveyed.

In Chapter 3 the main features of logic programming as a modelling tool are covered. The connections between a logic programming language Prolog and first order predicate logic are discussed. Then the two different readings of Prolog programs, declarative and procedural, are presented. An extension of pure Prolog is Definite Clause Grammars (DCG), a grammar formalism that is used to analyse and generate text and equations. The expressive power of DCG is shown to be adequate for analysing the textual form of word problems.

In Chapter 4 we describe how the model was constructed. We start the construction with a textbook analysis. Then we analyse the equations behind the textbook word problems. The equations are parsed with the DCGs to an ordered set of relations, operations and objects. We can not continue the construction before we have parsed the text to an ordered set of corresponding elements. Some elements that are found in the equation, objects or relations or operations, are missing from the text. We add the elements using general mathematical knowledge or specific knowledge from the problem domain. When we have added the missing elements we need to express all objects with the help of one specified object. This object is the unknown. Then, finally, we are able to construct a mapping between an equation and a word problem.

2. RELATED WORK - HISTORY AND MOTIVATION

Word problems have also been studied by some of the greatest men of science. "Such men as Descartes, Newton, Leibnitz, and Euler did not find it beneath their dignity to explain and illustrate at length the application of equations to the solution of word problems" [Poly-65]. They used mathematical word problems as a means towards a universal method for solving all problems. Although the quest for a universally perfect method has not succeeded, it has in many ways influenced science.

In this chapter we first investigate computer programs that are not intended to support education. Then we consider computer programs that have been planned to support education. Intelligent Tutoring Systems are shown to be the only area in AI research where AI is viewed only as a modelling tool. We consider in detail the Situation Problem-Solver, a computer program that was the first computer model for mathematical word problem-solving. Finally, we discuss the impacts of the computers and graphics calculators on modern mathematics curricula. We show that the use of these new tools enhances the status of word problems.

2.1 HISTORY OF WORD PROBLEMS

The capabilities for problem-solving and reasoning are generally taken as the principal factors of an intelligent system in AI research. The concept of an "intelligent system" has changed its meaning together with the evolution of computer programs. Usually these "intelligent systems" have been designed to a restricted problem domain e.g. to chess playing. The search for universal methods to solve a variety of different problems in different domains have inspired researchers in Artificial Intelligence from its early days. The first, and perhaps the best known work in this field is the GPS (General Problem Solver) model by

Newell and Simon [NeSi-63]. The GPS is a program that tries to simulate human thought. Newell and Simon argue that the free behaviour of human can be understood as a product of the finite and determined set of laws. The GPS model raises many philosophical questions [Marj-90]. However, the GPS "has led to generalised schemes of representation for goals and plans, methods for constructing discrimination nets, procedures for the control of tree search, pattern-matching mechanisms, and language parsing systems" [NeSi-76].

Word problems have also been used in the field of computational linguistics. Early implementations in the 1960's were often required to *understand* their inputs [Shap-87, p 134-136]. Bobrow wrote his dissertation "Natural Language Input for a Computer Problem-Solving System" in 1964 [Bobr-68]. In it he developed a system called STUDENT that solves high-school level algebra story problems given in a subset of English. The program uses pattern matching and well-chosen heuristics[3]. Bobrow developed techniques which allowed a computer problem-solving system to accept natural language. His main interest was in linguistic issues and not in educational nor didactic ones. His main motivation for choosing word problems as the domain was that the algebraic equations may be used to store the information needed to answer the questions in this context.

The STUDENT became famous partly because of its ability to solve most of the word problems found in a standard high-school algebra textbook, and partly from the use of the word "understanding" in his dissertation. He gives the following operational definition of word "understanding":

> *A computer understands a subset of English if it accepts input sentences which are members of this subset and answers questions based on information contained in the input. The STUDENT system understands English in this sense* [Bobr-68].

If we forget the definition, we may have to conclude that the STUDENT, a computer program, really understands English. Research about understanding is still a central theme in word problem-solving, e.g. in Polya's methods for solving word problems [Poly-65], in Kintsch and Greeno's problem-solving model [KiGr-85], in Greeno's application of this model to mathematics education [Gree-87], and in Reusser's cognitive simulation [Reus-90].

---

[3]E.g. in problems like : Mary is 4 years older than Susan..."years older than" is converted to "plus".

Charniak developed Bobrow's ideas and in 1969 implemented the CARPS system, which could solve calculus word problems at freshman-level. In reflecting on his work, Charniak noted that, despite its advances over the STUDENT, a very powerful calculus word problem solver will require a good deal of 'common sense' knowledge. In advancing this claim, Charniak established a new domain in AI, called the story understanding. One question in the field of story understanding has been left unanswered: What does it mean to understand a story, and how is it possible to determine whether a computer program was successful in this undertaking? [Shap-87, p.135].

One can argue, like Bobrow, that the above mentioned problem ceases to exist if we use word problems as the domain. This holds, if we think that to understand a mathematical word problem is to be able to solve that problem. Or more precisely, we can understand a word problem if we can construct the relevant equation(s). However, only a few common textbook problems force students to do in-depth semantic analysis. Thus it can also be argued that sometimes it is easier to solve a problem than it is to understand it:

> *The way text problems are formulated and how they work out can provide subtle hints to the problem solver which may let him accept a solution even if he does not understand it. To come up with a correct solution and be quite sure about it may not always mean that one understands it, even if the solution was inferred by several steps* [Reus-88].

Both the STUDENT and the CARPS transformed natural language into a set of simpler sentences, so-called kernel sentences that were equivalent in meaning[4]. These simple sentences were then transformed into a set of equations. Both of these systems used global information about the problem area, and the systems were able to ask for missing information from the user. However, Simon has stated that the STUDENT tried to translate the natural language problem *directly* into equations [Simo-83].

---

[4]Kernel sentences belong to the theory of transformational grammar (TG), a theory behind most of the works in computational linguistics (CL) in the 1960s'. The hypothesis of TG implies that "an understanding of syntax, or structure, of natural-language sentences can be arrived at on a solely grammatical basis, without considering the real world properties (e.g., meanings) of the terms being discussed" [Shap-87].

This statement is an oversimplification of these systems. To justify this statement, we have to look at the more recent work of word problem programs. Most of the early work prior to mid 1970's concerned computational linguistics. Novak implemented the ISAAC program in 1976. It solves college-level physics problems found in standard textbooks. The main difference between the STUDENT and the ISAAC is that the ISAAC first builds up an internal representation of the situation. This (physical) representation or "mental model" is constructed with the help of a special data bank, where the information of the basic physical objects is stored in the form of schemas. The internal representation does a lot of work, because it identifies the points where forces have to be equilibrated and therefore identifies which equations have to be set up. It can be said that the ISAAC in a restricted sense understands the problems it is given. But as Simon argues the ISAAC already knows all the physics it's going to know. To be able to really understand the world of physics the system must also be able to learn, or to create new schemas from the natural language [Simo-83].

Historical word problem solvers were mainly interested in computational linguistics. For our purposes two major facts are to be noted: Mathematical word problems form a strongly restricted subset of natural language, and an internal representation between an equation and a word problem is needed. Therefore it is possible to build programs that can analyse the textual form of most word problems found in standard textbooks. The internal representation of word problems is a good starting point for modelling the problem solving, too.

2.2 COMPUTER ASSISTED INSTRUCTION

The above mentioned programs had their origin in computational linguistics. Another branch of programs contains those programs that are originally planned to support education. Computer-assisted introduction/learning (CAI/CAL) started with *linear programs*. Linear programs were designed in late 1950's according to the behaviourist tradition with very little possibilities for individualisation. They simply proceeded step by step towards a goal, and accepted only right and rejected wrong answers. *Branching programs* were natural successors of linear programs in 1960's. They used pattern matching techniques to separate right and wrong answers [Nwan-90]. These programs were able to comment answers to users and use the answers to select an appropriate next step. The impact of these first trials can be seen in common teaching materials: linear programs have easily been

transcribed easily to ordinary textbooks and branching programs are applied in textbooks that use "programmed learning"[5]. Also the use of multiple-choice questions has been influenced by the technology that made them so easy to use [ScEd-89]. *Generative programs*, first implemented in the late 1960's, could provide something that was not possible from textbooks: they were able to generate teaching material. A number of *adaptive* systems were implemented to provide drill and practice in arithmetic, and to select problems at the level of difficulty appropriate to the student's overall performance [SlBr-82].

Different forms of CAI failed according to Nwana [Nwan-90] for the following reasons:

1. They attempted to produce total courses rather than concentrating on building systems for more limited topics.
2. They had severe natural language barriers which restricted users' interaction with them.
3. They had no "knowledge" or "understanding" of the subject they tutored or of the students themselves. Consequently they tended to assume too much or too little student knowledge, and they could not conceptualise so as to diagnose a students misconception with his/her own framework.
4. They were extremely *ad hoc*. Building tutoring systems was not recognised to be a non-trivial task - a task requiring detailed psychological theories of learning and mislearning. Anyone with a knowledge of computing attempted to build a tutor. Consequently there was little or no co-operation among educators, psychologists and computer scientists in the development phase of these tutors.
5. They tended to be static rather than dynamic. There was little experimentation with systems in order to improve them. Human tutors learn about their students and about the subjects they teach every day and so should machine tutors.

Basic types of CAI-programs, linear, branching and generative, were developed over 20 years ago. These programs had to be re-evaluated when micro computers were started to be purchased for schools in the1980's. The invasion of computers to school was mainly due to factors that were related to the overall social

---

[5]Textbooks that use the method of "programmed learning" comply with the following idea: "If you know the answer to this sub task then go to page nn, otherwise go to page mm" [Sein-74].

development. The schools did not request computers and at first they wondered what to do with them [Meis-87]. What the schools needed were educational programs. In Finland the government supported individual programmers. As a result plenty of programmers made the above mentioned mistakes. Usually only those ideas that were easy to implement evolved to CAI-programs. An attempt was made to reimplement our current curriculum on computers, but we should have investigated what computers can do to reshape education to answer the needs of modern society [ScEd-89]. No-one asked, or rather no-one answered the question: What parts of the curriculum needs to be taught with the help of computers? Thus the CAI programs stayed didactically on a very modest level [DuMa-87]. CAI-programs were basically computerised work-books. Word problems played no role in this CAI-episode.

School practice needs something what CAI-programs in 1980's could not offer. Will the answer be found in new menu driven programs with advanced user interfaces? Or are these components only syntactic sugar to unsolvable problems in CAI-paradigm? Nwana points out [Nwan-90] that CAI-programs do not know *what* to teach, *who* they teach and *how* to teach it. Clancey and Soloway argue [ClSo-89] that CAI architecture is inadequate to provide rich and robust learning environments. This was noticed already in early 1970's by researchers who applied AI in CAI systems. They started to design and implement ICAI (Intelligent Computer Aided Instruction) programs.

ICAI-programs have traditionally had the following three components in common [ClSo-89]:

1. *What to teach*:
   A model of the domain and an expert program that can solve problems in the domain.
2. *Who to teach*:
   A model of the student that identifies what the student understands.
3. *How to teach*:
   A tutoring model that can provide instruction to remediate misconceptions and/or present new material.

Wenger has suggested for a fourth component [Weng-87], a component whose importance is growing together with the possibilities of new technology:

4. *How to interact*:
   A model of user interface.

Sleeman and Brown's "Intelligent Tutoring Systems" [SlBr-82] gave a new acronym ITS to ICAI. Researchers who do not want to use the term "intelligent" use names such as KBTS (Knowledge Based Tutoring Systems), ATS (Adaptive Tutoring Systems), or KCS (Knowledge Communication Systems) [Nwan-90]. The terminology is as yet unestablished, e.g. term "learning environment" was first used in open ended systems or systems that support discovery learning like the LOGO language [Pape-80]. More and more often this term is used to emphasise the comprehensiveness of the program, or the term "learning environment" is used as a synonym for ICAI [ClSo-89].

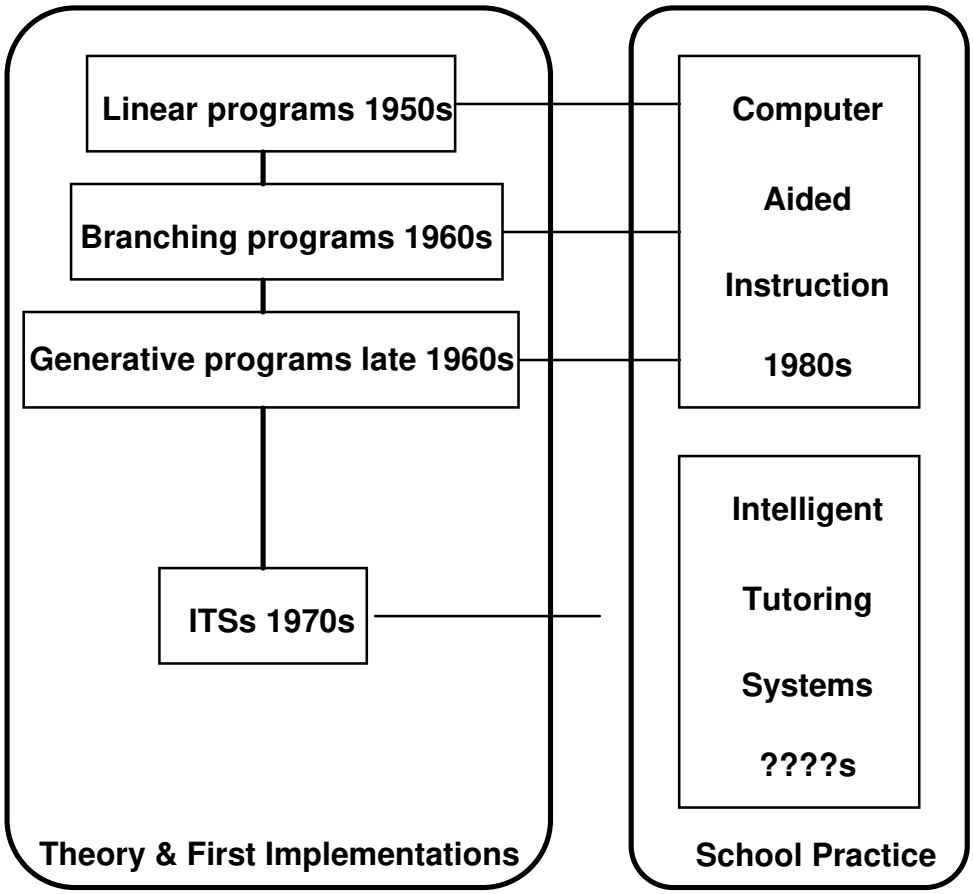

Figure 3. From CAI to ITS in theory and in (school) practice. Modified from [Nwan-90].

The learning environment as such is nowadays considered one of the most important topics to be studied in the educational sciences.

ITS programs have not yet reached the level of sophistication that is needed to satisfy the needs of modern education (Fig. 3). Building an ITS is a non-trivial task. It demands expertise at least from the following disciplines: AI, educational theory and psychological models of student and expert. It has been suggested that "ITS research will grow. This is because, apart from their practical needs, the area appears to provide an excellent test-bed for theories to AI scientists, educational theorists and cognitive psychologists" [Nwan-90]. One practical need behind the development work of CAI and ITS has been to reduce the high cost of human tutors and thus to develop systems for automatic education [AnBo-90].

ITS	**Domain**	**Reference**
ATDSE	Basic subtraction	Attisha & Yazdani (1983)
ARITHMEKIT	Basic subtraction	Brown (1983)
ALGEBRALAND	Algebraic proofs	Brown (1985)
BUGGY	Basic subtraction	Brown & Burton (1978)
DEBUGGY	Basic subtraction	Burton (1982)
EDSMB	Basic multiplication	Attisha & Yazdani (1984)
FITS	Basic fractions addition	Nwana(1990)
GEOMETRY Tutor	Geometry proofs	Anderson et al. (1985)
INTEGRATION Tutor	Basic integral calculus	Kimball (1982)
LMS	Basic algebra	Sleeman & Smith (1981)
PIXIE	Basic algebra	Sleeman (1987)
QUADRATIC Tutor	Quadratic equations	O'Shea (1982)
SIERRA	Basic arithmetic procedures	Vanlehn (1987)
SPIRIT	Probability theory	Barzilay (1985)
WEST	Basic arithmetic skills	Brown & Burton (1978)

Table 1. A reasonably comprehensive list of ITSs in mathematics education according to [Nwan-90].

The aim to develop systems for automatic education may never succeed:

> *I personally do not believe that computers compete with human teachers in any real sense, or that they can or should replace people in instructional settings. Teaching as social act simply involves too many dimensions*

beyond the exact processing of information for which computers are made and at which they can excel [Weng-87].

We are using the methods of AI to model and thus to better understand the problem-solving process of word problems. Our aim is not to replace the teacher but to build a model to be used by the teacher. To advance towards this goal we need to study the history of ITS, because ITS is the only area in AI research where AI is only viewed as a *modelling tool* [ClSo-89].

Mathematical word problems are not included in a comprehensive list of 43 ITSs [Nwan-90, p. 272]. The fifteen which are related to mathematics education are listed in Table 1. We might add some new ITSs whose domain is mathematical word problems. The algebra Word Problem Tutor by Singley et al. [SiAn-91], the HERON by Reusser et al. [ReKä-90], and the ANIMATE by Nathan et al. [NaKi-92]. The builders of these systems strove for two goals. The first one was to develop a pedagogical tool to be used in classrooms and the second one was to explore some fundamental psychological problems surrounding word problem-solving.

The Algebra Word Problem Tutor is based on a recent work on tutoring basic algebraic manipulation skills [SiAn-91], whereas the HERON has been developed by Reusser for several years [Reus-87a, Reus-87b, Reus-88, Reus-90]. The ANIMATE presents a representational base to the problem-solving process by giving situation-based meaning to the equations through computer animation [NaKi-92]. The ANIMATE and the SPS are both based on the van Dijks and Kintch's theory of discourse processing [KiDi-78]. We shall closer look at Reusser's previous work below.

Our main purpose is to build a model of word problem solving and not to build an ITS. However, we think that the building of an ITS must start in the conceptual analysis of the situation. Also some kind of a model about the situation must exist before a robust ITS can be built. The TEACHER is to be understood in this sense. In fact, the TEACHER is at the moment only a model of the domain, an expert program that can solve problems in the domain. A tutoring model of word problem-solving is evolving [HaKi-89, ReKä-90, NaKi-92]. User modelling easily considers man to be deterministic and mechanistic [Järv-93]. A lot of work must be done before a real ITS in this domain can be built.

2.3 SITUATION PROBLEM-SOLVER

In the following we shall outline the main features of Reusser's Situation Problem-Solver. The SPS was one of the first attempts to combine the methods used in the field of AI with the research of mathematics education. Both the SPS and this work share the same interest, namely how to transform a mathematical word problem into an equation. The SPS has a strong theoretical background in the discourse analysis of van Dijk and Kintsch [KiDi-78] and in the process model of problem-solving by Kintsch and Greeno [KiGr-85]. We argue that the discourse analysis and the temporal reasoning cause some theoretical limitations to the model.

Reusser implemented in the mid 1980's the Situation Problem-Solver (SPS), a computer simulation model of understanding and solving word arithmetical problems [Reus-87a, Reus-87b, Reus-88]. The SPS uses the general theory of text comprehension developed by Kintsch and van Dijk [Hoik-90]. The basic idea behind the theory is that the comprehension process can be decomposed into components, some of which are manageable at present, while the others can be put aside until later. Because of the complexity of the natural language, two important components are excluded from the model. The model will be concerned only with semantic structures. Furthermore, comprehension always involves knowledge use and inference processes. The model does not specify details of these processes [KiDi-78].

If we consider general theories of problem-solving [Elor-74], or the specific theories on mathematical word problems [Poly-45, Poly-65, Lewi-81], we notice that some important features are missing if we apply the above mentioned theory to mathematical word problems. Of course, the details of human inference processes are unknown and complicated[6], but the use of knowledge is an essential feature in general problem-solving [Chom-72, Elor-74]. The other missing feature is the inference process from the mathematical text (syntax) to its real meaning (semantics) (Table 2). This is the first phase in solving a word problem, a phase

---

[6]The difficulties concerning human inference processes are discussed in [Bode-90] and [Sear-92].

that Polya calls *Understanding of the Problem* (Fig. 11) We will discuss this phase in Chapter 5.

The SPS also uses Kintsch and Greeno's processing model that explicitly deals with both text-comprehension and problem-solving aspects of word arithmetic problems [KiGr-85]. Reusser's studies in word problems were part of the broader research project "on theoretical relations between text comprehension, situation comprehension, and mathematization in solving word arithmetic problems, and on the teachability of these issues" [Reus-87b]. Reusser has continued his work on the psychological processes of how students understand and solve mathematical word problems [Reus-88]. These works have been behind the development of the HERON system [ReKä-90].

The SPS has five modules: 1.) The Text Comprehension module parses and encodes the Textual input into the Textbase. 2) The Situation Comprehension module goes beyond the text and concentrates on time-dependent or temporal reasoning. 3) The Mathematization module puts aside all those elements that are not relevant to the operational gist of the episode. All elements are presented in the same order than the children tend to recall a problem episode i.e. in a chronological order. This phase, the Mathematical Problem Model, is a link between the semantic and the numerical representation of the problem situation. 4) The Calculation Module uses different counting strategies for the first graders. 5) The Answer Interpretation module uses the result of The Situation Comprehension module, the Episodic Problem model, in answer generation (Appendix 2).

Reusser has done a profound work on refining and extending the ideas of Kintsch and Greeno in understanding the word problem-solving process. Reusser has also demonstrated that a computer model which is intended to be a cognitive simulation model is "an exciting tool both for the study of the phenomena, the generation of theories as well as for the design of experiments" [Reus-87b]. He has used this knowledge in the development of the HERON system. The HERON is "a computerised didactic tool for facilitating and fostering self-direct and comprehension-based learning and problem-solving" [ReKä-90]. The HERON helps the student to solve word problems by providing support in text comprehension and in building mathematical problem models. The HERON can handle a fixed number of problems that have been implemented in the system when it has been developed. These problems vary from two simple sentences and three figures to problems with over 300 words and over ten different figures.

Reusser has concentrated on the psychological issues of text comprehension and problem-solving process. He has not considered some of the factors that are crucial for the model in detail. One of these factors is parsing and encoding the textual input and the other is time dependent reasoning. In the following we consider time dependent reasoning because time is a central component in the SPS model and the SPS model has been used as starting point of our work.

The roles of time in problem-solving are [Kahn-89]:

- Storing and retrieving time-varying, context sensitive data
- Detecting temporal relationships from time-ordered data
- Reasoning about change and processes
- Representing dynamical systems
- Reasoning about states and context.

Related research areas are:

- database management
- mathematical modelling
- symbolic knowledge representation
- temporal inferencing

The time problem is not the unique property of word problems. It is a general problem of all those intelligent systems that reason about the changing world. In the SPS the time problem is equivalent to the problem of reasoning about change. A fundamental assumption in constructing intelligent systems is that the problem solver or reasoning component has an internal model or representation that captures the essential features of the problem world. This component is called Textbase in the SPS. The problem solver reasons by manipulating the elements of this internal world. The enterprise of constructing formalisms for expressing real-world entities in computer-usable internal models is called knowledge representation. The difficulties associated with time and change are problems in the available knowledge-representation techniques for constructing internal models that successfully and efficiently capture key properties of a dynamic, evolving world [Kahn-89]. In AI the problem dealing with changing propositions

has been called the frame problem[7]. The simplest solution to this problem invokes the frame assumption: The only facts that change are those facts that are explicitly told to be changed.

In the SPS the frame problem is not present. The SPS only uses a static number of pre-defined problems. The problems are given to the SPS as a list of micropropositions and not as sentences of problem text (Table 2).

Problem text	**Input**
Walter tends to lose everything.	(PRESENT) (EQUAL X WALTER) (EVERYTHING)
He got twelve beautiful shells from Julia last week as a birthday present.	(PAST) (LAST WEEK) (EQUAL Y JULIA) (GET X Y) (12) (SHELL) (BEAUTIFUL) (AS-BIRTHDAY-PRESENT)
Today Walter has only eight shells left.	(PRESENT) (TODAY) (HAVE-LEFT X) (8) (SHELL) (ONLY)
How many shells did Walter lose?	(PAST) (LOSE X) (HOW MANY) (SHELL)

Table 2. The natural language version of one of the problems that the SPS is able to analyse and the actual input format for the computer program.

---

[7]Original definition comes from the situation calculus, where the main problem is to handle the situation where something changes. In situation calculus, a change in state results in the removal of logical statements that are no longer applicable, and in the inserting of new logical statements that are permitted in the new state [McHa-69].

Many experiments in Artificial Intelligence have been centred around simple-looking problems, because they "give us, for the smallest initial structures, the greatest complexity, so that one can engage some really formidable situations after a relatively minimal diversion into programming" [Mins-68]. Reusser also used simple looking problems to investigate the inner structure of word problems:

> *Anyone who wants to teach children how to understand and solve mathematical word problems, needs to know by which processes or strategies, and mediated by which mental representations, mathematical information is abstracted from verbal problem statements* [Reus-90].

2.4 A PROGNOSTIC VIEW OF WORD PROBLEMS

The author of this study has acted as a teacher and a teacher trainer for over ten years. In the 1980's the calculators changed the teaching practice in schools. The time spent to numerical calculations diminished and arithmetically more demanding questions could be set. The computers then came to schools. In this section we investigate how the computers have changed mathematics education.

The most recent innovation used in high-schools is graphics calculator. As a teacher, it is easy to predict how graphics calculator will change mathematics education. In this section, we investigate these possible consequences.

Lewis found out that the experts performance in solving elementary equations was not sharply different to less skilled solvers [Lewi-81]. The experts make mistakes and, surprisingly, they don't just look at problems and write down the answers. Lewis concludes that people don't necessarily improve just by doing something more frequently [Lewi-81].

Lewis and Matz [Matz-82] use the phrase "algebra problem-solving", when they studied man's ability to solve algebraic equations. In this study we are mainly interested how word problems are "translated" into equations and not how these equations are solved. However, the thoroughly examined area of solving equations

can be seen as valuable background information. Many high-school algebra errors of solving equations are due to the conceptual changes involved in moving from arithmetic to algebra:

> *Syntactically similar expressions differ semantically because equal sign, concatenation, plus and minus have multiple meanings which are distinguished only by parsing rules and context. Even though equality is mathematically symmetrical, cognitively and pragmatically it is not. The bi-directionality of algebraic processes injects a flexibility into algebraic problem-solving which, in the eyes of some students, makes the activity more complicated* [Matz-82].

There are two specific reasons that support the choice of word problems from the didactic point of view. First, word problems are much more difficult for students than comparable algebraic problems [HaKi-89, Reus-90]. Second, the significance of word problems is rapidly growing. The Curriculum in Finnish schools is going to change from the beginning of the school year 1994-1995 and there is a strong emphasis in problem-solving and word problems. The general trend to use more problem-solving and less drilling of basic mathematical manipulation in education is clearly seen also in the new curriculum for the International Baccalaureate high schools [Anon-93]. Another argument for the growing significance of word problems can be found in the advancement of computers. Never in the history of mathematics education has no single invention opened so many challenges and demands for educators than the computers. Computers have influenced education in three ways:

- Mathematics, and the way mathematicians are working has changed.
- From the above fact and from the availability of software that already can perform many mathematical tasks, it follows that the whole curricula have to be re-evaluated.
- Mathematics education has gained new possibilities by using computers to support education.

IB has mandated the use of calculators in mathematics examinations from May 1994 and allowed the use of calculators with graphical display (IB World, December 1992). Also the Finnish national board for matriculation examination[8]

---

[8]The board (Ylioppilastutkintolautakunta in Finnish) sent on 11.8.1993 a note to headmasters of the Finnish high schools. The note admits several interpretations. The most probable

has allowed the use of powerful graphic calculators in the Finnish matriculation examination from March 1994. We will take a closer look what implications this new technology has for the way mathematics is taught and which items in the tests are to be replaced. We first examine properties of one of the most powerful calculators that IB accepts, TI-85™.

TI (Texas Instrument) 85™ is in fact not a calculator but a small graphics computer with 32 kilobytes of RAM. It has a 21×8 characters display. Its equation-solving capabilities allow solving any variable in an equation, solving 30 simultaneous equations, finding roots of polynomial up to the 30th order and solving differential equations up to ninth order. It solves calculus problems both numerically and graphically; finds first and second derivatives, integrals, minima, maxima and inflection points. It has special editors for handling vectors, matrices, lists, strings and complex numbers. It performs statistical analysis for one or two variables, is able to use 7 different regression models and is able to display the data also graphically. In addition it can handle parametric equations and polar functions.

Many of the above mentioned features are not necessary in Finnish matriculation examination but nearly all of them come into use in Higher Level IB-mathematics examination. The use of graphics calculators in examinations has not yet started and therefore there exists no research in this area. However, the subject was thoroughly examined and discussed in a seminar for European IB-teachers in mathematics (Lüneburg Germany, March 1993). Three main questions were addressed by Chief Examiners in IB Mathematics as they pondered the equality issues of the availability and use of calculators in IB schools around the world:

1. Which mathematics content is most appropriately tested with a calculator?

2. How do test items on calculator-based tests differ from those in tests not allowing the use of the calculator?

---

interpretation is that the same calculators can be used in Finnish matriculation examinations and IB-examinations.

3. How can test items be constructed so that they may be calculator-inactive, calculator-neutral or and/or calculator-active[9]?

These questions were still found to be mainly unanswered. However, some findings were stated:

1. *Mathematics objectives* commonly cited as appropriate for testing with a calculator include:
   (a) exploration of number patterns,
   (b) use of guess-and-check strategy for problem-solving,
   (c) processes of hypothesis formulation and verification.

2. Test questions should not measure *calculator skills only* but should be based on the mathematics curriculum.

In the following we shall show how difficult it is to find appropriate test questions in present school books. But first, what are the benefits of calculator-based items and what kind of items should they be? Some of the answers that were mentioned in the Lüneburg conference were:

(i) Students should be enabled to concentrate on strategic approaches to problems without getting tangled up in memorised computational algorithms.

(ii) The calculators should enhance the validity of items by permitting the use of more realistic data or numbers, and thereby allowing problem-solving situations to be more akin to those found in practice.

(iii) The use of calculators should enable the amount of time required to complete individual questions to be better controlled.

(iv) All items on a test should not require the use of calculator. The items should also be constructed in such a way that using a calculator might affect the performance of the student, positively, negatively or not at all.

---

[9]Calculator inactive problems are those for which there is no advantage (perhaps even a disadvantage) in the use of the calculator. Calculator neutral problem can be solved without calculator but calculator *may* be useful. Calculator active problems require the use of calculators for their solutions.

(v) Items should emphasise the testing of problem-solving rather than computational skills.

How many changes should be made to present test items? To evaluate this question we chose two recent test papers: the Higher Level IB-mathematics paper, which was thoroughly evaluated at the Lüneburg conference, from spring 1992 (Appendix 3) and the corresponding higher level paper of the Finnish matriculation examination from spring 1992 (Appendix 4).

The Finnish examination consists of 10 problems that must be solved during 6 hours. Five of those ten problems had parts a and b. The student should choose either part a or part b, i.e. there were 15 problems altogether. Five problems (Problems 2, 3, 6a, 7, and 8) can be solved directly with TI-85™ (Calculator active problems). Considerable part of the problem can be solved in seven cases (Problems 1, 4b, 5a, 6b, 9b, 10a, and 10b) with TI-85™ (Calculator active problems). Little or no use of the calculator (Calculator neutral problems) is in three cases (Problems 4a, 5b, 9a). Problem 9a is an abstract geometrical item, other calculator neutral problems are word problems (4a and 5b and also calculator neutral part of the problem 6b). No problem was considered to be calculator inactive.

The IB-examination in Higher Level mathematics has two parts. For the first part the students have 1.5 hours and they have to solve 20 questions. Correct answers give full marks, i.e. the students do not necessary have to show their reasoning. One third of the points could be gained by using the TI-85™ (questions 4, 9, 11, 14, 18 and 19). Also the questions 1, 2 and 8 are more easily handled by TI-85™ than by a conventional calculator. Part two of the test has four optional topics of which the student has to solve one. These are mainly abstract mathematical problems and thus mainly calculator neutral (Appendix 3).

The above mentioned analysis of the use of TI-85™ in the tests has been carried out without considering the programming capabilities of the calculator. The calculator has quite advanced programming language and enough memory (32k bytes of RAM) to store the programs. Students can copy these programs from other calculators or from MacIntosh™ or IBM™ compatible computers. IB regulations allow the use of stored programs (IB World, December 1992). Students can invent these programs themselves or they can copy them from the

manual or from books that have been published for supporting the use of TI-85™ in mathematics education.

The new mathematics syllabus for IB has been published to be used for examinations from May 1995 onwards [Anon-93]. Higher Level Mathematics has five major parts: Fundamental mathematics, Probability, Functions and Calculus, Matrices and Vectors, and one optional topic. Students may use a graphics calculator with the above mentioned features. Test questions are to be such that student who do not have the possibilities to use graphics calculator[10] are not disadvantaged. This situation demands some special properties for the test questions. They must test more deeper and abstract mathematical understanding than just mechanical manipulation of mathematical formulae[11]. Lower level courses, Mathematical Methods and Mathematical Studies in IB and the lower level mathematics of the Finnish Matriculation Examination, do not handle very abstract mathematical problems. A considerable part of the curriculum of these subjects consists of word problems [Anon-93]. The availability of graphics calculators most probably will increase the number of word problems both in the lower and higher level mathematics.

---

[10] Many IB-schools are in developing countries. These schools have problems even in getting enough ordinary calculators for the examination, because slide rules are no longer allowed from May 1994 onwards.

[11] In the latest (September 1993) matriculation examination of the Finnish higher level mathematics the four easiest problems were calculator active problems (Appendix 5). This situation causes some problems because graphical calculators can be used in the next examination (March 1994).

3. PROLOG AS A MODELLING TOOL

In this chapter we investigate which theoretical features does a logic based model have and how the implementation of the model with Prolog is intertwined with the abstract modelling process. In section 3.1 we show that Prolog has its roots in mathematical logic. The Prolog clauses have two different readings, declarative and procedural. The declarative, or descriptive, reading makes the model easy to write and easy to interpret. We show that the procedural reading of Prolog programs is analogous with mathematical analysis. In section 3.2. we investigate the properties of Definite Clause Grammars (DCG). We show that the expressive power of the DCG is sufficient to define a reasonable sub-set of the language of word problems.

3.1. PURE PROLOG AS A MODELLING LANGUAGE

The TEACHER prototype has been implemented by a logic programming language Prolog (PROgramming in LOGic). The Prolog's syntax is that of the first-order predicate logic formulas written in clause form, and further restricted to Horn Clauses [Brat-86]. The language based on Horn Clauses is as expressionable as the first order predicate calculus [Walk-87]. Horn clauses are also called Definite Clauses [PeWa-80]. Horn Clauses take the form:

A if B_1 and B_2 and ... and B_n

The corresponding Prolog-clause is of the form

A :- B_1, B_2, ...,B_n.

This clause can be read, and executed, as a procedure of a recursive programming language, where A is the procedure head and the B_is are its body [StSh-86]. In other words this clause can be read *declaratively*

A is true if B_1 and B_2 and ... and B_n are true.

The clause can also be read *procedurally*

to solve (execute) A, solve (execute) B_1 and B_2 and ... and B_n , or
to do A, do B_1 and do B_2 ... and do B_n [Kowa-87].

The declarative, or descriptive, reading of Prolog programs is an interesting feature from the point of view of a modelling process. We are able to write clauses that tell us how modelling is supposed to happen in general terms and at the same time we get an executable Prolog program i.e. we can divide a problem into parts that no longer have to be *functionally* dependent. We are able to construct recursive models that share similar relational features. If we can clearly express how the model is logically constructed, we then have also an executable model on Prolog.

We can notice the difference between the problem solving process and the model constructed with Prolog. Our model has a definite logical structure that has a correspondence in reality. We also have some features in the model and in reality that correspond to each other. The model tells us how we get from a certain feature to another more general feature. But the model tells us this in logical terms only. The model does not intend to show how our brains or minds process the knowledge to get from one feature to another. The model is to be understood in this sense, i.e. *declaratively*.

> *A logic program is a set of axioms, or rules, defining relationships between objects. A computation of logic program is a deduction of consequences of the program. A program defines a set of consequences, which is its meaning. The art of logic programming is constructing concise and elegant programs that have the desired meaning* [StSh-86].

To understand better the declarative interpretation of our model we have to take a closer look at the basic constructs used in Prolog. These constructs have their origin in first order predicate logic. There are three statements in Prolog:

- Facts

- Rules
- Queries

Facts are Horn clauses with no Tail.	(A if True).
The procedural reading of a fact is	to do A, do nothing.
Rules are Horn clauses with Head and Tail.	(A if B_1 and B_2 and ... and B_n).
Queries are Horn clauses with no Head	(? if B_1 and B_2 and ... and B_n).

The only data structure in Prolog is *logical term*. Term can be *constant*, *variable* or *structure*. Constants are either *atoms* or *integers*. Atoms begin with a lowercase letter or with a special character. Variables start with an uppercase letters or with the underscore _ [12]. Structure is a compound term, i.e. a term that has several components. The components themselves can also be structures. For example the structure

triangle(isosceles(top_angle, relation(6,*),base_angle)

may refer to an isosceles triangle whose top angle is six times greater than its base angle. Interpretation has to be stated explicitly. In this structure triangle is the (principal) *functor* of the term.

It is not plausible, and it is naive to believe that we as humans process information like Prolog machines. However, it is worth noticing how the Prolog programs are executed *procedurally*. The procedural meaning of Prolog programs is based on the principle of *resolution* [Robi-65]. The resolution principle may be presented briefly in the following form [Walk-87]:

$$\frac{A_1 \vee A_2 \vee \mathtt{K} \vee A_m \vee \neg B \qquad B \vee C_1 \vee C_2 \mathtt{K} \vee C_n}{A_1 \vee A_2 \vee \mathtt{K} \vee A_m \vee C_1 \vee C_2 \mathtt{K} \vee C_n}$$

[12] Variables that need not be named are called *anonymous variables*. Usual convention of referring to anonymous variables is to use the single underscore _.

In other words, the premises $A_1 \vee A_2 \vee \mathtt{K} \vee A_m \vee \neg B$ and $B \vee C_1 \vee C_2 \mathtt{K} \vee C_n$ imply that the statement $A_1 \vee A_2 \vee \mathtt{K} \vee A_m \vee C_1 \vee C_2 \mathtt{K} \vee C_n$ is true[13]. If $m = 0$ and $n = 0$ then we have two premises $\neg B$ and B and no conclusion. In logical terms we have

$$\frac{\neg B \qquad B}{\textit{Contradiction}}$$

Resolution-based theorem proving is behind all reasoning in Prolog-programs. For instance if we have

$$\frac{f \qquad s \vee \neg f}{s} \quad \text{this is identical to} \quad \frac{f \qquad f \rightarrow s}{s} \quad \text{which is written} \quad \frac{f \qquad f :\!- s}{s} \quad \text{in Prolog-notation.}$$

Thus, from *fact* f and from the rule: *fact* f *implies statement* s we conclude that *statement* s *is true* (Modus Ponens).

Resolution is based on *unification*. "Unification is the basis of most work in automated deduction, and the uses of logical inference in Artificial Intelligence" [StSh-86]. A unification algorithm is presented in Fig 4. Unification, as defined in many Prolog implementations for efficiency reasons, does not exactly correspond to unification as defined in logic. However, from a practical point of view this approximation to unification is quite adequate [Brat-86].

---

[13] This is a shorthand notation because B and $\neg B$ may include variables that initialise when B and $\neg B$ unify. This procedure may have an affect the terms A_i and C_i.

Input: Two terms T_1 and T_2 to be unified
Output: θ, the most general unifier (mgu) of T_1 and T_2, or a failure.
Algorithm:
Initialize the substitution θ to be empty, the stack to contain the equation $T_1 = T_2$, and failure to *false*.
while the stack not empty and no failure do
pop $X = Y$ from the stack
case
X is a variable that does not occur in Y:
substitute Y for X in the stack and
in θ add $X = Y$ to θ
Y is a variable that does not occur in X:
substitute X for Y in the stack and
in θ add $Y = X$ to θ
X and Y are identical constants or variables:
continue
X is $f(X_1, \ldots, X_n)$ and Y is $f(Y_1, \ldots, Y_n)$
for some functor f and $n > 0$:
push $X_i = Y_i$, $i = 1, \ldots, n$ on the stack
Otherwise:
failure:= *true*
if failure, then output *failure*;
else output θ

Figure 4. A unification algorithm [StSh-86, p.71]

Has this definition of resolution any counterparts in human thinking? The word resolution is of Greek origin and its Latin equivalent is *analysis*. The most famous description of analysis is that of Pappus (300 AD.):

> *In analysis, we start from what is required, we take it for granted, and we draw consequences from it, and consequences from the consequences, till we reach a point that we can use as starting point in synthesis. For in analysis we assume what is required to be done as already done (what is sought as already found, what we have to prove as true). We inquire from what antecedent the desired result*

> *could be derived; then we inquire again what could be the antecedent of that antecedent, and so on, until passing from antecedent to antecedent, we come eventually upon something already known or admittedly true. This procedure we call analysis, or solution backwards, or regressive reasoning* [Poly-65].

The two definitions for word "resolution" are closely related. The greatest difference is that Prolog uses pure computation [StSh-86], whereas human beings use different (geometrical) heuristics [Niin-83].

3.2. EXTENSIONS OF PURE PROLOG

The declarative meaning of pure Prolog programs makes the program, or the model we are building, easy to write and easy to understand. For practical reasons of executional efficiency pure Prolog programs are not possible. All Prolog implementations offer *system predicates*. For example, for arithmetic comparison there are system predicates `>` and `<` that directly call the underlying arithmetic functions of the computer. In addition, all Prolog implementations make use of *meta-logical predicates*[14] to enhance the expressive power of the programs. For example, the term *var(Term)* tests whether the given term is at present an uninstantiated variable [StSh-86].

A special predicate to control and to prevent backtracking is called *cut* (written as !). If inserting a *cut* destroys the correspondence between the declarative and the procedural meaning of the program, the cut is called *red cut*. Otherwise it is called *green cut*. Use of *extra-logical predicates* causes side-effects and declarative interpretation of the program is not possible. Extra-logical predicates are concerned with I/O, or accession of data and manipulation of the program, or interface with the underlying operating system [StSh-86].

---

[14] The name meta-logical predicates denotes to the fact that these predicates are outside the scope of the first-order predicate logic.

We have above briefly described the main features of logic programming as a modelling language. To summarise:

- Logic programs may be expressed in a language that is easy to understand.
- Logic programs are build to handle relational features.
- Logic programs can be understood both declaratively and procedurally (with the above mentioned exceptions).

These features may be applied to modelling processes in general. In the following we shall introduce a special property of Prolog programs that helps us to model the word problem-solving process. The TEACHER prototype analyses and generates mathematical word problems. The TEACHER prototype also analyses and generates equations. We need a tool to analyse and generate both the artificial language of textbook examples, and the artificial language of equations.

Prolog has been designed in the early 70's to natural language processing by Alain Colmerauer and Robert Kowalski. In 1978 Colmeraurer used a grammar formalism to assist in making a translation between English and French [Colm-78]. Grammars that may be analysed by Prolog are called *logic grammars*. An extension of pure Prolog, Definite Clause Grammars , briefly DCG [PeWa-80], is a grammar formalism that is build in many Prolog implementations. The TEACHER prototype makes use of DCG in two modules. Here, DCG is initially used to analyse and generate word problems and secondly to analyse and generate equations.

In order to understand logic grammars, we need to make explicit some general features of grammars. If we want to make an accurate definition of a *language*, we define a collection of *rules* called *grammar*. The rules of a grammar define whether the given string of words is a valid (generated) sentence of the language. "In addition, the grammar generally gives some kind of analysis of the sentence, into a structure which makes its meaning more explicit" [PeWa-80]. It is easy to define grammars with Prolog. The analysis and synthesis are forms of inference that are executable by Prolog's inference mechanism. The grammar, being simply an axiom system or a collection of rules, is declarative in nature, and hence the grammar is easy to create and understand [McCo-87].

Type 0 or *General Rewrite Grammar*
Productions of these grammars have no limitations. It can be shown that languages of this type are exactly those languages that a Turing machine is able to detect [HoUl-79].

Type 1 or *Context-sensitive Grammar*
There has to be at least as many non-terminals on the right hand side as there are on the left hand side of the productions.

Type 2 or *Context-free Grammar*
There is only one non-terminal on the left hand side of the productions.

Type 3 or *Finite State Grammar*
Productions take the form $X \rightarrow aY$ or $X \rightarrow Y$
where X and Y are non-terminals and a is a terminal.

Figure 5. Chomskian hierarchy.

Logic grammars inherit their symbols and notations from linguistics [Chom-57]. A grammar is defined as a quadruple (VN,VT,P,S), where VN is the set of non-terminals, VT is the set of terminals, P is the set of productions and S is the starting non-terminal. The alphabet V of the language is an union of the terminals VT and non-terminals VN. Chomsky defined four different types of grammars. The most general one is of type 0 and the most restricted one is of type 3. In this *Chomskian hierarchy* (Fig. 5) the higher level grammar is able to produce all the structures that the lower level grammar is able to produce. In addition the higher level grammar also produces other kind of structures. All productions take the form

$$X \rightarrow Y$$

where X and Y are strings of alphabets V.

Type 2 or Context-free Grammars, CFG, and Prolog clauses are analogous. Prolog clauses have only one term on the left hand side of the rule and many or none on the right hand side of the rule. Definite Clause Grammars[15] are normally used as level 2 Grammars. In fact DCG is a generalisation of Context-free Grammars. CFG-grammars are usually defined and represented in notation based on Backus-Naur Form [PeWa-80]. A CFG-grammar is defined by stating production rules between the terminals and non-terminal of the language. CFG cannot adequately describe natural language, nor even many programming languages. DCG overcomes this inadequacy by extending CFGs in three ways (Fig. 6).

1. *DCGs provide for context-dependency in a grammar, so that the permissible forms for a phrase may depend on the context in which that phrase occurs in the string.*

2. *DCGs allow arbitrary tree structures to be built in the course of parsing, in a way that is not constrained by the recursive structure of the grammar; such tree structures can provide a representation of the "meaning" of the string.*

3. *DCGs allow extra conditions to be included in the grammar rules; these conditions make the course of parsing depend on auxiliary computations, up to an unlimited extent.*

Figure 6. DCG in an extension of Context Free Grammars [PeWa-80].

After these extensions a DCG is normally still a Context Free Grammar. "Definite clause grammars are essentially context-free grammars augmented by the language features of Prolog" [StSh-86, p. 260].

We are defining a subset of natural language, the language of some word problems found in standard textbook(s), in DCG-notation. We are able to write the language of these word problems in DCG-notation (Fig. 7). This task may seem difficult or even impossible. However, it has been argued that word problems form an

[15]The name Definite Clause Grammars comes from logic. Prolog is based on Horn Clauses (Section 3.1) that are also known as Definite Clauses.

semantically poor subset of natural language [Bobr-68]. We shall show in Section 4.3 that with the help of a couple of concise DCGs we are able to analyse and generate all the word problems (and more) found in one standard Finnish textbook [PaVo-89]. These simple DCGs are easily translated into other languages, in the TEACHER prototype we have translated them into English (Fig. 10).

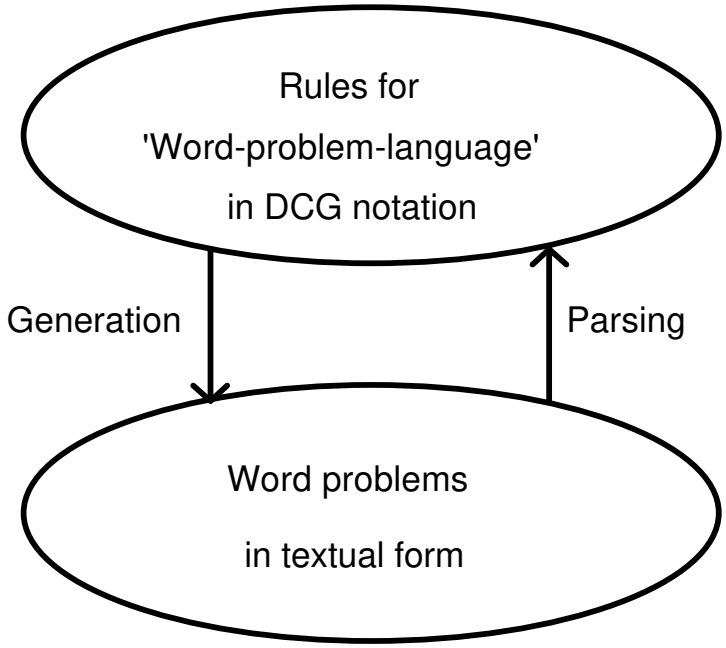

Figure 7. The language of word problems may be written in DCG-notation. This notation is used to generate, and to analyse, other word problems.

To clarify how the DCGs are to be understood we take a closer look to DCG-notation. The productions of CFG may be written with DCG. DCG is also an executable Prolog program. Therefore we can use it to identify and to generate sentences that comply with the rules of the grammar. CFG-rules have the following form:

non-terminal-symbol $\rightarrow$ body.

where "body" is a sequence of one or more items separated by commas. Each item of the body is either a non-terminal symbol or a sequence of terminal symbols. The syntax of DCG is presented in Fig. 8 using the notation of its own rules. This is a modified version of [ClMe-84]. Our version considers also the backtracking

mechanism of Prolog i.e. Fig. 8. presents an executable Prolog program that is tested with LPA-MacPROLOG™.

```
grammar_rule            --> head, ['-->'], body, ['.'].
head                    --> non_terminal.
head                    --> non_terminal, [','], terminal.
body                    --> element_of_the_body, !.
body                    --> body, [','], body.
body                    --> body, [';'], body.
element_of_the_body     --> ['!'].
element_of_the_body     --> ['{'], Prolog_goals,['}'].
element_of_the_body     --> non_terminal.
non_terminal            --> prolog_atom(X).
terminal(X)             --> list(X).
list([])                --> [].
list([Head|Tail])       --> prolog_atom(Head), list(Tail).
prolog_atom(X)          --> [X],{atom(X)}.
```

Figure 8. The syntax of DCG. Modified version of [ClMe-84].

The two first rules state that DCG rules are identical with CFG rules. Third rule allows both non-terminals and terminals to be written in the head of the rule. The expressive power of the grammar is enhanced and the grammar is able to present context dependent properties (Rule 1 in Fig. 6).

The body of the rule consists of one single element or of several elements separated with commas. The element may also be separated with a semicolon. Semicolon is interpreted as a disjunction. The use of semicolons is, however, not recommended because large grammars then have side-effects that are not easily eliminated. An element of the body may be cut '!'. There are some reservation against cut, but efficiency is improved when cut is used properly.

An element of the body may also be a Prolog goal or several different Prolog goals separated with commas. Each of these goals may call another Prolog program (Rule 3 in Fig. 6). These goals are not satisfied when the logic grammar is executed. They are satisfied together with other goals when execution continues.

Elements of the body may also be terminals or non-terminals. Non-terminals are Prolog atoms and terminals are empty lists or lists of Prolog atoms [ClMe-84]. Thus the translation[16] of these grammars to Prolog becomes easier [StSh-86]. The features that are relevant for modelling mathematical word problems are summarised in Table 3.

	Property of the program	**Feature of the model**
Logic programs	may be expressed in a language that is easy to understand.	Models can be made transparent.
	are build to handle relational features.	Models can be made bi-directional.
	can be understood both declaratively and procedurally.	Two separate meanings of the word "understanding".
	support advancing through abstraction levels.	Models may contain abstraction levels.
Prolog programs	have been originally designed for natural language processing.	Word problems may be handled efficiently.
	have generally a built-in tool, logic grammars, for language analysis and generation and at the same time an elegant way to represent grammars.	Word problems, and equations, may be analysed and generated and the grammars may be easily understood and enlarged.

Table 3. The advantages of logic programming in general modelling and the advantages of a special implementation of logic programming, Prolog , in modelling mathematical word problems.

---

[16]For reasons of efficiency DCGs are translated to difference lists [StSh-86]. This can be done automatically with a help of a short algorithm and the efficiency may be enhanced from $O(n^2)$ to $O(n)$ [ZhGr-88].

4. THE CONSTRUCTION OF THE MODEL

In the two previous chapters we have investigated the properties of logic programming as a modelling tool, and the relevant theories that relate to word problem-solving. The model we are building between a word problem and an equation shall be developed on the basis of this theoretical groundwork. The constructive part of this study is presented in this chapter.

The Situation Problem-Solver, (SPS, Section 2.3), has been used as an inspiring example for the present work. Some of the names of the different abstraction levels are inherited from the SPS. However, the SPS and our model look the problem-solving from different perspectives, and thus the abstraction levels are not analogous.

The first step in building a word problem solving model is to analyse ordinary textbook problems. Chapter 4.1 tries to find answers to questions such as: What are the typical problems that are used as a first introduction to mathematical word problems that are to be solved using equations? Can we find common features of word problems to group them into different types?

In Chapter 4.2 we analyse the structure of equations behind different word problem types. Can we find common features of these equations? Or can we even find universally applicable features of equations behind word problems? Furthermore, can we find analogous features from the original word problems and is there a mapping between these elements?

In Chapter 4.3 we use the answers of the above mentioned questions to construct the intermediate abstraction levels of the model.

4.1. THE LANGUAGE OF WORD PROBLEMS AND THE PROBLEM DOMAIN

We analyse one widely used Finnish textbook [PaVo-89] and show that the results can be applied to another famous Finnish textbook [HeKu-88]. Before the textbook analysis, we shortly describe how the properties of logic programming (Table 3.) can be applied to word problem solving.

The main advantage of using logic programming as a modelling language is the bi-directionality of the model. The traditional, LISP-based models of word problem-solving processes have a strong functional flavour. The starting point of the analysis has, quite naturally, always been the word problem itself. From the days of the STUDENT system [Bobr-68], word problems have been used in natural language processing. Also Reusser's SPS-model is based on the discourse analysis and text comprehension of van Dijk and Kintsch.

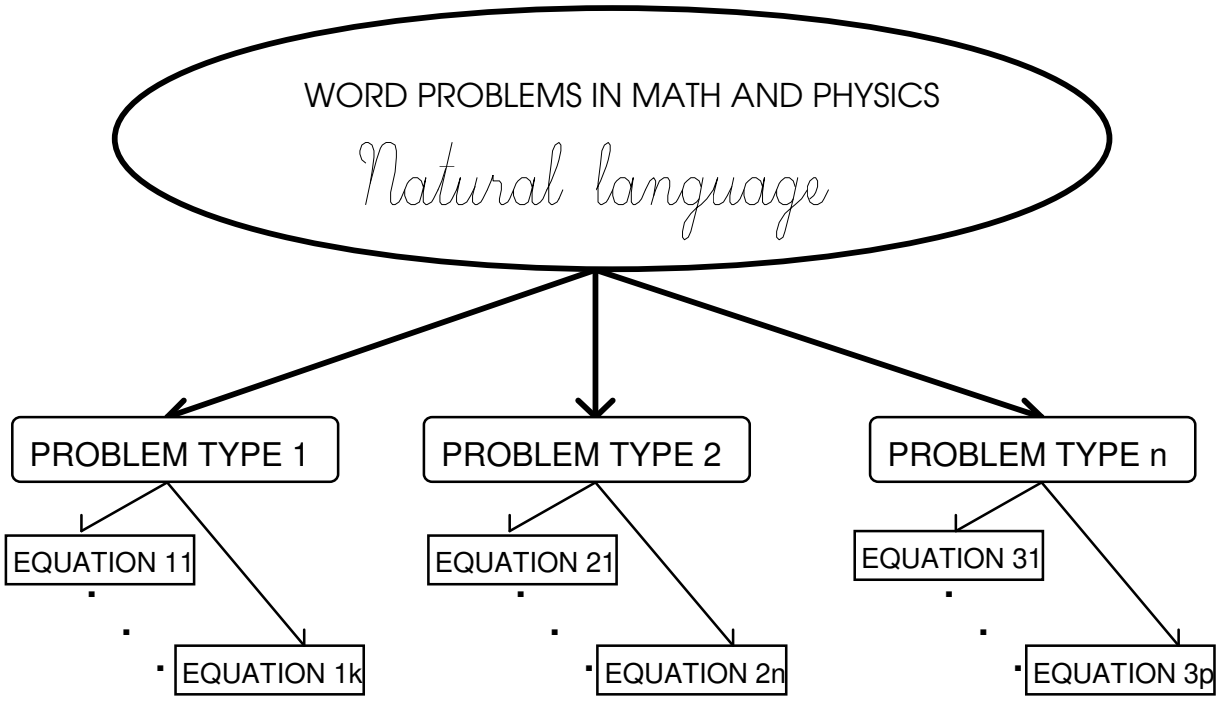

Figure 9. The starting point of traditional models for word problem-solving has been word problems and the problems concerning natural language understanding.

An overview of the traditional models is shown in Fig. 9. The greatest problem is the diversity of natural language. In the SPS this problem is bypassed by giving the word problems to the SPS as a list of micropropositions obtained from the sentences of the problem text (Table 2). Deriving these micropropositions from the text is in fact an essential feature of word problem-solving. We can not leave this important phase without consideration.

We can not analyse natural language with computers, but we want to analyse word problems presented in textual form with computers. We have solved this anomaly by creating an artificial language for word problems. The grammar of this language is written in DCG-formalism. In fact, different problem types (Fig. 9) need their own DCG-grammars. Textbook word problems are usually presented using different categories or problem types. Therefore it is possible to define grammars for different problem types by analysing ordinary textbooks.

Time and change are essential components of situation problems i.e. problems where states, events and actions play the main role [Reus-87b]. We do not consider these problems, because it is still an open question, how to represent time-dependent data in databases (Section 2.3). The situation problem solving is important for very young children but the algebraic structure of the equation is more important for older students "who know very well the problem types with their associated mathematical reasoning strategies" [Reus-89].

We are concentrating on ordinary word problems of secondary school textbooks, where the problems of time and change are normally bypassed. The theoretical background of the TEACHER model is quite different from the traditional models. In the TEACHER model we are interested in those mathematical word problems where the explicit goal is to construct an equation and solve it. This is the most common type of word problems in secondary and high school textbooks in Finland. One can argue like Reusser:

> *Many textbook math problems are not intellectually challenging because they are formulated as semantically poor, disguised equations instead of thinking stories or situation problems, which do not allow students bypass thorough semantic analysis in order to solve them* [Reus-88].

However, the semantically poor word problems are widely used in mathematics and physics textbooks. We wanted to build a simplified test environment for modelling the problem-solving process. In doing this we chose that part of the

curriculum [Anon-82] where mathematical word problems are to be solved by first forming an equation. This is another difference between the traditional models and the TEACHER model. The traditional models have been mainly interested in elementary arithmetical word problems [BrLa-84, CoVe-85, Reus-87b]. The TEACHER model concentrates on the first introduction of algebraic word problems. This part of the curriculum is extremely important because of the conceptual changes involved in moving from arithmetic to algebra, especially the maturation of the concept of equality [Matz-82]. This subject is taught in Finland at the end of the 7th school year to 14 year-old students.

The most widely used textbook in Finland in the beginning of 1990's [PaVo-89] was chosen as a starting point. First a problem-analysis was carried out. Most of the examples were mechanical i.e. of the form "solve the equation..." and only 37% of the problems were of the form "build an equation and solve it". The word problems could quite easily be divided into four categories (Table 4). The textbook encourages students to solve word problems by first determining its problem type or problem category. The first category, literal representation of an equation, is a simple one and it is not taken into further consideration. The remaining three categories, age problems, triangle problems, and money-sharing problems, form the test-problem group for the TEACHER prototype. The test-problems are all, surprisingly, of the same, and indeed semantically poor, type.

The same problem types were found in another well-known Finnish textbook [HeKu-88]. Surprisingly, the grammar for the triangle problems developed on the basis of [PaVo-89] (Fig. 10), was adequate to analyse also the triangle problems in [HeKu-88].

Problem type	Typical problem	%	Equation
Literal representation of an equation	Form an equation and solve: A certain number that is multiplied by fourteen is equal to the product where the sum of the member and one is multiplied by four	20/32 = 60%	14x = 4(x+1)
Age problem	Father is three times older than son. If you add the age of the father and the age of the son you get 60 years. Form an equation and solve it to find how old the father and the son are.	3/32 = 10%	x+3x = 60
Triangle problem	The top-angle of an isosceles triangle is four times greater than the base-angle. Build an equation and solve it to find the degrees of the angles.	3/32 = 10%	x+x+4x = 180
Money sharing problem	Jim, Jane and Bill were thinning carrots. Jim worked two times as much as Bill and Jane worked three times as much as Bill. Build an equation and solve it to find how the 366 mk pay should be shared.	6/32 = 20%	x+2x+3x = 366

Table 4. The different problem types of a widely used Finnish textbook AHAA7 [PaVo-89] for 7th graders. The problems are from that part of the curriculum, where mathematical word problems are to be solved by first forming an equation.

4.2 DCG-GRAMMARS

In this section we start to discuss the model construction together with implementational issues. Our main aim is to build a model in terms of logic programming. The main emphasis on the implementational issues can be found in this section. The final part of this model (Section 4.3) is implemented problem type specifically i.e. new problem types can not be added automatically. We are describing the model and the prototype declaratively. A further research question

is: are we also able to implement the rest of the model , Section 4.3, using the same DCG formalism as we have done in this section?

In the previous section we saw that word problems can be grouped into four different types. The second step is to analyse the equations behind the textbook word problems.

Before we start to analyse equations we shortly describe the properties of Prolog programs (Table 3.) and especially the properties of LPA-MacPROLOG 3.5 [ClMc-88]. The language of the word problem and the language of equations must first be transformed into a list of tokens. Tokens are basic elements, e.g. words or numbers, of the text or of the equation. It should be noted that no capital letters may be used in Prolog. Capital letters start the names of variables in Prolog.

Equations and word problems are given in strings and not as lists of tokens, e.g.

> "In triangle ABC the angle A is two times as big as the base angle B and the angle C is six times as big as the angle B. Form an equation and solve it to find the angles of the triangle."

or "x + 2x + 3*2x = 180"

The predicate

tokenise(String, List).

transforms a given string , i.e. a word-problem or an equation, into a list of tokens. First a string is decomposed into letters, digits and characters such as question marks, comparison operators (>, <) etc. Capital letters are transformed to ordinary letters and fill chars such as space, line brake etc. are bypassed. The single characters are further combined to different kinds of tokens such as words, numbers, or special characters. If for instance the variable String is instantiated with "x + 2x + 3*2x = 180" :

tokenise("x + 2x + 3*2x = 180"**, List)**,

then the variable List is instantiated with: List = [x,+,2,x,+,3,*,2,x,=,180].

Before we continue with this list representation of an equation we reiterate that the basic idea behind the TEACHER system is to look at the mathematical word problem from the equation (Fig. 1). The word problems we are interested in are of the form "build an equation and solve...". Therefore the equation can be represented as a set of objects, operations and relations that all have a corresponding entity in the word problem.

The equations in Table 4 have all an integer value at the right hand side of the equation, and the left hand side is a sum or difference of two or three objects. The equations can be represented in the following general form:

object1 operation object2 operation object3 **equals** total_value

or

object1 operation object2 **equals** total_value

In the case of the triangle problem in Table 4, the objects are the angles of a triangle and the operations tell us to sum these objects together. The left hand side of the equation forms a combined object (sum of the angles of a triangle) and the right hand side of the equation is another object (180 degrees). Thus the equation can be seen as an equivalence relation between two mathematical entities, or objects:

LEFT-HAND-SIDE-OBJECT **equals** RIGHT-HAND-SIDE-OBJECT

The OBJECTs can be decomposed into an ordered set of objects and operations (left hand side) or the OBJECTs may represent a simple value (right hand side). The decomposed objects have, in equations that are derived from mathematical word problems, relationships with each other. In other words the equation can be decomposed into a set of objects, operations and relations between these objects. This formal or context free model may be applied to the underlying equations of the word problems found in Table 4.

We are now ready to construct the first transformation of the model. The above mentioned analysis of an equation is performed using DCG. The starting non-terminal is '*equation'*. The predicate

parse(List, Structure, Grammar_rule).

builds a syntactic structure of the given word problem. The variable List is the original equation transformed into a list of tokens and Grammar_rule is the starting non-terminal (Fig. 8) of the grammar. If, for instance, the variable Grammar_rule is *'equation'* and the list is [x,+,2,x,+,3,*,2,x,=,180] then the Structure is:

obj(obj(x),obj(2*x),obj(3*2*x),oper[+,+]) , obj(180)

In other words, we have two objects obj(180) and another object that consists of three sub-objects, obj(x), obj(2*x) and obj(3*2*x), and operations, oper [+;+], that bind these sub-objects together. We call this structure *the context free model*.

This general model of an equation, or the context free model (Appendix 1), can be constructed without knowing the word problem it will represent. The transformation between an equation and the context free model is implemented using logic grammars. In grammar formalism we have implemented the usual way of writing an equation from text. For instance, if we have the following equation:

"x+x+60+2x=180"

we get the left-hand-side-objects:

obj(x) , obj(x+60) and obj(2x)

or some other possible combination e.g.

obj(x) , obj(x), obj(60), and obj(2x)

All operations between the objects are '+'. The former combination of objects has a representation in triangle problems, but the latter does not have a representation in any of the problem types in Table 4. We may also want to express, which object we want to process. This is done by using parentheses:

"(x)+ (x+60) + (2x) = 180".

The transformation between an equation and the corresponding context free model is not arbitrary. We are transforming equations into a set of objects that have corresponding entities in a word problem. Therefore one object, obj(x), has to be the unknown. Other objects have to be represented using this unknown. In other words, there has to be a relation between different objects. For instance:

obj(2x) is 2*obj(x).

If we want a more sophisticated model, we only have to recursively decompose the objects into smaller objects or entities. This recursive decomposition is needed in high school physics problems, or more generally in problems where we have simultaneous equations. But in our model the illustrated decomposition is sufficient. In fact, relations such as: top-angle is four times greater than the base-angle, may be represented in a form of a simple equation. Thus also the examples in Table 4 are formally solved by solving simultaneous equations. However, this is not the way we introduce the solution process to students. The traditional way of teaching students to solve these problems starts from noticing the objects, relations and operations from the textual form of the problem. The traditional way of teaching to solve mathematical word problems is analogous with Polya's way of introducing the subject (Fig. 11). This discussion ends the first transformation of the model, the transformation between an equation and the context free model.

The next transformation starts from the textual form of the word problem. Our ultimate aim is to find a mapping between an equation and a word problem. We have described how to transform an equation into an ordered set of objects, relations, and operations. The transformation from an equation to the above mentioned general form, context free model (Appendix 1) in the TEACHER prototype, is implemented bi-directionally by a logic grammar. Logic grammars are also used in the transformation of a mathematical word problem bi-directionally into a formal representation that we call *the textbase*. A logic grammar of the triangle problems is in Fig. 10. The textbase is composed of certain entities - objects, relations and operations - that are derived from the text.

```
triangle((P,R,Q))                  →    problem(P), condition(R), ['.'], question(Q), ['.'].

condition((R1,R2))                 →    condition(R1), [and], condition(R2).
condition(R)                       →    relation(R).

problem(spec_object(triangle, Name))          →    [in, triangle, Name], !.
problem(spec_object(triangle))                →    [in, triangle].
problem(spec_object(isosceles_triangle)) →         [in, an, isosceles_triangle].

relation(rel(A, C2, '*', B))   →    object(A), [is, C1, times, as, big, as], object(B),
                                    {make_num(C1, C2)}.
relation(rel(A, equals, B))    →    object(B), [is, as, big, as], object(A).
relation(rel(A, C,'+', B))     →    object(A), [is, C degrees, bigger, than], object(B).
relation(rel(A, C,'+', B))     →    object(B), [is, C degrees, smaller, than],
object(A).

object(top_angle)              →    [the, top, angle],!.
object(base_angle)             →    [the, base, angle],!.
object(Name)                   →    [the, angle, Name].

question(question)             →
                  [form, an, equation, and, solve, it, to, find, the, angles, of, the, triangle].
```

Figure 10. A simplified DCG grammar to be used with Prolog both to *recognise* and to *parse* the sentences it defines, and to *generate* all its sentences.

Word problems are given in strings and not as lists of tokens, e.g.

> "John is four times as old as Mary. If you subtract the age of John from the age of Mary you get 60 years. Form an equation and solve it to find out how old John and Mary are."

The first phase of the *lexical analysis* is done with the same predicate that is used to tokenise equations:

tokenise(Word_problem, List).

The predicate transforms a word-problem, into a list of tokens. The predicate

parse(List, Structure, Grammar_rule)[17]

builds a syntactic structure of the given word problem. The variable List is the original word problem transformed into a list of tokens and Grammar_rule is the starting non-terminal (Fig. 8) of a fitting problem type (Table 4). If, for instance, the Grammar_rule is *'triangle'* then the program uses a DCG-grammar to parse triangle problems. A simplified DCG-grammar for triangle problems is presented in Fig. 10. This grammar may be enlarged by adding more detailed rules for the language. In fact only a small fraction of the possibilities of DCG-grammars is used (Section 3.2).

It is possible to build a parse tree of the word problem using DCG-formalism. The DCG syntax presented in Fig. 8 is able to build the logical form of the word problem. In Fig. 8 the non-terminal symbols are Prolog *atoms*. If we allow the non-terminal arguments be Prolog *structures* (Fig. 10) then DCG can be used for building syntactic structures instead of logical forms [McCo-87]. We are using DCG-grammars to produce the mathematical meaning of the word problem. With the help of DCG-grammar we are able to separate data and condition, to separate the different parts of the condition and to state the relations between the data. This phase is analogous with Polya's scheme (Fig. 11). The same grammar formalism is used to generate all the word problems the system is able to recognise.

---

[17]The predicate parse is the following:

```
parse(List, Structure, Grammar_rule):-
    X=..[Grammar_rule, Structure, List,[]),
    call(X).
```

If the first clause, with the built-in predicate =.., is true then a new term X=Grammar_rule(Structure, List, []) is constructed. This new term is not a syntactically acceptable clause. With another built-in predicate *call* , whose argument is the goal X, this problem can be solved.

Word problems are said to be semantically poor [Bobr-68, Reus-88]. Our findings verify this statement. All triangle word problems found in two Finnish textbooks for seventh graders [PaVo-89, HeKu-89] have the above mentioned grammar. In other words, the TEACHER is able to recognise and analyse all these problems. In addition, the TEACHER can generate more (relevant) problems than can be found in these textbooks. Enlarging TEACHERs ability to solve different types of problem can be performed by inserting grammars into the system.

4.3. A MAPPING BETWEEN AN EQUATION AND A WORD PROBLEM

We have analysed the structure of an equation. The equation, underlying a basic word problem, has a definite structure. We have shown that it is possible to transform this structure into a context free model. Nearly analogous structure (textbase) can be found from the textual form of word problems. The final phase of the construction is to find out if there exists a mapping between the textbase and the context free model.

It is possible to define, with DCGs, the language of ordinary mathematical word problems. Using this grammar we are able to transform a word problem into a textbase. The transformation between word problem and the textbase is done with the predicate[18]:

```
from_text_to_textbase(Word_problem, Textbase):-
    tokenise(Word_problem, List),
    parse(List, Structure, Grammar_rule).
```

For instance, if variable Word_problem is given the value:

> "John, Paul, and Mary delivered advertising leaflets. Paul worked six times as much as John and Mary worked five times as much as Paul. How should the £285 pay be shared?"

---

[18]The variable Grammar_rule is instantiated with all implemented starting non-terminals of the DCG-grammars. Only one of them is possible in this prototype.

then the result of the lexical analysis, the parse tree of the word problem, is the Textbase:

textbase(object(john,paul,mary),relation(paul,6,*,john),relation(mary,5,*,paul),
object(tot_value(285), question(all_objects)).

However, the parse tree is not easy to read because it is one single Prolog structure. We want that the textbase is built during the processing in a more readable form. For this we apply an uncommon property of Prolog programs.

A special property of Prolog programs is the program's ability to modify itself. New clauses may be asserted and existing clauses may be retracted during the runtime[19]. An excessive use of these facilities may obscure the meaning of the program. The resulting program may become difficult to understand and difficult to explain [Brat-86]. We demonstrate these features by asserting only those facts and rules that logically follow from the program. Adding such clauses will not affect the meaning of the program, since no new consequences can be derived from it [StSh-86]. The TEACHER asserts facts (data and conditions) and relations between the data that are obtained by the above mentioned lexical analysis. The TEACHER also deduces new facts from the obtained relations. In textbase we now have objects, relations between these objects and the unknowns.

We are now able to present an instance of the textbase of the above mentioned word problem in the following form:

fact(object(john, paul, mary)).
fact(rel (paul, 6, *, john).
fact(rel(mary, 5, *, paul).
fact(total_value(285)).
question(john, paul, mary).

We have now shown how to construct two abstraction levels, textbase and context free model. The context free model is a collection of objects, relations, and

---

19 Using extra-logical predicates *assert* and *retract* it is also possible to introduce set-predicates that produce sets as a solution. This is one example of second-order programming [StSh-86].

operations. Some corresponding components are missing from the textbase. The additional components can be derived from the knowledge that has been learned previously. The textbase is enlarged, we call the new abstraction level *the enlarged textbase*, if necessary with the help of general mathematical knowledge

a equals b if and only if a is 1*b,

or context specific knowledge

the sum of the interior angles in any triangle is 180°.

In the above mentioned problem type no general mathematical knowledge is needed, but we need to use the following context specific knowledge: the sum of the money that each person gets is the total amount of the money to be shared. In other words, the extra knowledge is : the operations [+,+] bind the sub-objects john, paul and mary together to a larger object, and this new object is equivalent with the total amount of money to be shared. The extra knowledge is inserted to the textbase with the predicate:

enlarge_textbase(Textbase, Enlarged_textbase).

The next phase is to choose the appropriate variable. We then can express all objects using the unknown. We call this abstraction level *the context sensitive level*. It contains information that is needed to build an equation. The following heuristic is used to select the variable: *Try to avoid structures where the unknown is in denominator*. Therefore Mary and Paul can't be selected as unknowns. The unknown will be John, or in fact John's share. The link between the enlarged textbase, and the context sensitive model is the predicate

from_structure_to_model(Enlarged_textbase, Context_sensitive_model).

An instance of the context sensitive model of the previous example is

variable(john).
rel(paul, 6, *, john).

```
rel(mary, 5, *, 6, *, john).
objects(john, paul, mary).
operations_between_objects([+,+]).
object(tot_value(285)).
```

Building a context sensitive model is more or less analogous to phase 2, devising the plan, in Polya's scheme (Fig. 11): the connection between the data and the unknown are presented explicitly. Related problems are problems where the total amount of money has to be shared. Choosing the unknown is done with the described heuristic.

The predicate

from_model_to_model(Context_sensitive_model, Context_free_model).

transforms the context sensitive model to relations between two objects that are equal, we call this abstraction level *the context free model*. The objects may contain sub-objects and operations between the sub-objects. An instance of the Context free model of the previous example is:

object(x) + object(6x) + object (5*6x) *equals* tot_value(285)

Here all the connections to the meaning of the word problem are set aside and only those features that are left are essential to form the equation. The last phase is to transfer the context free model to the equation. This is done with the predicate

from_model_to_equation(Context_free_model, Equation).

The last phase is trivial if we are looking at the problem-solving process from word problems to equation (Fig 1). But our starting point was in fact the equation and not the word problem. We have described in Section 4.2 how to transform an equation to a context free model. This is done with the above mentioned predicate where variable Equation is instantiated and the other variable is uninstantiated:

```
from_model_to_equation(Context_free_model, Equation):-
   tokenise(Equation, List),
   parse(List, Context_free_model, equation).
```

The predicates between each pair of two consecutive levels, e.g. enlarge_textbase(Textbase, Enlarged_textbase)., can be used in two ways: to generate the Enlarged_textbase from the Textbase, or to generate the Textbase from the Enlarged_textbase. Combining all the above mentioned predicates under one predicate, we have the TEACHER prototype, without the user interface module:

from_text_to_equation(Word_problem, Equation):-
from_text_to_textbase(Word_problem, Textbase),
enlarge_textbase(Textbase, Enlarged_textbase),
from_textbase_to_model(Enlarged_textbase, Context_sensitive_model),
from_model_to_model(Context_sensitive_model, Context_free_model),
from_model_to_equation(Context_free_model, Equation).

If variable Word_problem is instantiated (given) then the resulting equation is formed together with the abstraction levels, the Textbase, the Enlarged_textbase, the Context sensitive model, and the Context free model. If the variable Equation is instantiated (given) then the abstraction levels are built. There are usually many possible Textbases that can be built from one equation. Each of these Textbases is used to generate all the possible word problems that the DCGs can generate[20], e.g. the grammar in Fig. 10.

In this chapter we have shown how the TEACHER prototype was constructed. At the same time we have constructed a logic based model for word problem-solving. We call this model the TEACHER model. It is not easy to separate the abstract modelling process and the implementation with Prolog. Therefore the result of this construction is called the TEACHER system that contains both a model and a prototype.

---

[20] The program generates also Finnish word problems. This may be avoided by selecting the language. The selection is part of the user interface module of the TEACHER .

5. THE BI-DIRECTIONAL SYSTEM

In Chapter 4 we constructed the TEACHER model. In this chapter we investigate the properties of the TEACHER system i.e. the model and the prototype. The main predicate of the TEACHER prototype, from_text_to_equation(Text, Equation), is a relation between the textual form of the word problem and the equation. Other predicates describe the relations between different abstraction levels of the TEACHER system (Fig. 2, Appendix 1). Prolog programs are basically relational and therefore we are able to consider the problem-solving process from the student's view, from text to equation, and from the textbook's view, from equation to text. In fact, if we are using Prolog as a modelling tool we can not, strictly speaking, separate these two views (Fig. 1). In this chapter we are studying the applications of the system.

5.1 THE TEACHER AS A WORD PROBLEM GENERATOR

We have created an artificial language for word problems (Section 4.2). Or rather, we have invented a method by which the creation of the word-problem language is possible. We use different grammars for different problem types (Fig. 9). Enlarging a grammar of a problem type means we can analyse more word problems from the same problem domain. By changing the lexical part of the grammar we can change the language, in this version English and Finnish. Using analogous method, textbook analysis and DCG-formalism, we can widen the scope of the problems that the system is able to analyse. The system is entirely bi-directional. In this section we investigate the transformation from an equation to the textbase to word problems.

As we have seen in Chapter 2, the history of mathematical word problems is intertwined with the basic problems of natural language understanding. Traditional models of word problem-solving systems (Fig. 9) look the problems uni-directionally from text to equation. If we want to build a computer environment,

where true word problems are to be solved, we have first to overcome the manifold problems of natural language. The problem is obviously unsolvable if we do not restrict the natural language to a simplified subset. This is what Bobrow did in his STUDENT system. Bobrow showed that

> *The STUDENT system, which accepts as input a comfortable but restricted subset of English, can be used to express a wide variety of algebra story problems* [Bobr-68].

Bobrow used simple pattern matching heuristics to build his famous system. The grammar of the text that was acceptable to the STUDENT was hidden in these pattern matching statements.

From the context free model of the equation we are able to construct a group of word problems. The TEACHER first checks which problem types are possible (Appendix 1, Table 4). The problem type may be, for example, "Triangle problem", if there are three objects, the operations between these objects are "+" and there is a natural number 180 on the other side of the equation.

When TEACHER knows the possible and selected context , the system is able to construct a context sensitive model (Appendix 1). In the present version, we use fixed expressions e.g. the name of a triangle is ABC and the angles are A, B and C. When special knowledge of a context and general knowledge are taken off from the context free model, we get the textbase (Appendix 1).

The textbase is a collection of objects, relations between these object, and operations that combine these objects to larger objects. The entities of the textbase have a representation in the text. The textbase is a combination of exactly those entities that are relevant from the point of view of the problem-solving process. It is possible to find these entities in the word problems we are interested in, because these problems are ordinary textbook problems:

> *The general characteristics of textbook problems: well-defined with one solution which the teacher already knows; the solution is obtainable with one's own recourses; calculations working out evenly indicate being on the right track; confinement to relevance and non-ambiguity: everything that is relevant to the solution is stated in the text, and everything that is stated is relevant; the explicit problem question is always present and highly informative; all problems are solvable* [Reus-88].

The transformation between the textbase and the textual form of word problem is implemented by logic grammars. From the textbase the TEACHER generates a group of word problems to a certain problem area. There is a specific grammar for every problem type, one for triangle problems, one for age problems and one for money sharing problems in the present prototype version. Each of these grammars defines unanimously the language the TEACHER prototype is able to understand and generate. The grammars are easy to interpret and easy to enlarge. So we may broaden the system to generate more lexical variants from a single textbase. We can also implement new grammars for new problem types. Thus we are able to enlarge the system in a restrained manner.

To summarise: If we give the TEACHER prototype an equation as an input, the system generates a group of word problems to every possible problem type (Table 4). If we do not want to restrict the number of word problems in any way, the TEACHER generates all those problems that can be analysed by the prototype. All of the generated problems are understandable and may be found in standard English or Finnish textbooks. Even if we do not consider some trivial lexical variants, the number of the word problems generated by the present preliminary version may be counted in tens or hundreds.

5.2. THE TEACHER AS A MODEL OF WORD PROBLEM-SOLVING PROCESSES

The construction of the logic based model started from an equation and from a textbook analysis simultaneously. This first phase gave two abstract levels, textbase and context-free model as a result. A new level, the enlarged textbase, had to be added to maintain the correspondence between objects and relations. For a similar reason some of the objects or relations had to be taken off or rearranged. The final level to be constructed was the context sensitive model. In this section we shall show that the result of the construction is, quite surprisingly, analogous with Polya's way to teach word-problem solving.

First, we shall outline the meaning of the words "problem" and "problem-solving". The basic characteristics of a problem are universally agreed:

- The desire to attain some *goal.*
- The *means* by which this goal is pursued.
- *Unawareness* of the process, how (or by which combination of means) the goal can be reached.

Eloranta [Elor-74] has comprehensively discussed different problem-solving methods and heuristics. The above mentioned definition is also in accordance with Polya's definition of problem-solving:

> *To have a problem means: to search consciously for some action appropriate to attain a clearly conceived, but not immediately attainable, aim. To solve a problem means to find such action* [Poly-65].

Lewis has studied the specialist skills in the domain of algebra. To solve an algebra problem means that one must employ two kinds of knowledge [Lewi-81]. First, the solver must know an adequate set of correct operations. This corresponds to the *means* of the definition of a problem. Second, the solver must know how to *select* an appropriate operator to use at given juncture. This corresponds to the *action* of the definition of problem-solving and also to *the combination of means* of the definition of a problem. So, we can find all the basic principles of problem-solving in algebraic problems. Descartes tried, and failed, to reverse this process. Here is a rough outline of the scheme that Descartes expected to be applicable to all types of problems [Poly-65]:

- First, reduce any kind of problem to a mathematical problem.
- Second, reduce any kind of mathematical problem to a problem of algebra.
- Third, reduce any problem of algebra to a solution of a single equation.

"When a high school boy solves a "word problem" by "setting up equations", he follows Descartes' scheme and in doing so he prepares himself for serious applications of the underlying idea" [Poly-65]. Polya himself considers the general

problems from the same mathematical perspective as Descartes. Polya argues that in any problem there must be

- an *unknown*
- something *given* (the data)
- a *condition*, which specifies how the unknown is linked to the data

These concepts can be found in the textbase of the TEACHER. Question, *question(A,B,C)* in the textbase, is a collection of all the unknowns of the problem. Objects, *object*s(X,Y,Z) in the textbase, refer to the given features of the problem. This is, however, not a straightforward thing. We should rather say that what is given is the explicit *values* of the objects, and those objects that have no values are unknown. In the prototype there are just identical entities in objects(X,Y,Z) and in question(X,Y,Z). Thus it is better to state that in a mathematical word problem we have some objects, of which some are unknown, and that we have some conditions that link these objects together. These conditions are represented in the textbase with *relations* between the objects and with *operations* that combine these objects together to a larger object. In fact we have operations between objects, where some objects are unknown and some are expressible by known objects or by the relations of these known objects.

Although great scientists like Poincare and Einstein have been interested in heuristics, the real renaissance of heuristics in mathematics began only after Polya [Elor-74]. Eloranta states that the Polya's heuristics are probably (and unfortunately) not widely used in the mathematics teaching.

> *The "word problems" of the high school are trivial for mathematicians, but not so trivial for high school boys or girls or teachers. I think, however that a teacher who makes an earnest effort to bring Descartes' advice, presented in the foregoing, down to classroom level and to put it into practice will avoid many of the usual pitfalls and difficulties.* [Poly-65]

In the following we shall show that the TEACHER model quite closely follows Polya's guidelines (Fig. 11) and Descartes' scheme presented by Polya [Poly-65]. Our model is nearly analogous with Descartes' scheme. To emphasise this aspect we use the following notation. Polya's own words [Poly-65], printed in italics, are used to explain some features of the TEACHER model.

HOW TO SOLVE IT

UNDERSTANDING THE PROBLEM

First. *You have to understand the problem.*
What is the unknown?
What are the data?
What is the condition?
Introduce a suitable notation.
Separate the various parts of the condition.
Can you write them down?

DEVISING THE PLAN

Second. *Find the connection between the data and the unknown.*
Do you know a related problem?
Look at the unknown.
Here is a problem related to yours and solved before.
Could you use it?
Could you use its results?
Could you use its methods?
Could you think of other data appropriate to determine the unknown?
Could you change the unknown or the data, so that the new unknown and the new data are nearer to each other?
Did you use all the data?
Did you use the whole condition?

CARRYING OUT THE PLAN

Third. *Carry out your plan of the solution, check each step.*
Can you see clearly that the step is correct?
Can you prove that it is correct?

LOOKING BACK

Fourth. *Examine the solution obtained.*
Can you check the result?
Can you check the argument?

Figure 11. Polya's guidelines for solving word problems [Poly-45].

First of all, the student should not start doing a problem before he has understood it. It can be checked to a certain extent whether the student has really understood the problem. He should be able to repeat the statement of the problem (objects(A,B,C) in the textbase)*, point out the unknowns of the data* (question(X,Y,Z) in the textbase)*, and explain the conditions in his own words* (relations and operations in the textbase)*. If he can do all this reasonably well, he may proceed to the main business.* In other words, to be able to construct a

textbase is a sufficient condition to proceed. The textbase represents the notes that are written down when the student tries to understand the problem (Fig. 11). If we consider the same problem as in Section 4.3. i.e. "John, Paul, and Mary delivered advertising leaflets. Paul worked six times as much as John, and Mary worked five times as much as Paul. How should the £285 pay be shared?" then we have:

The Textbase	**The notes of a student or of a teacher**
fact(object(john, paul, mary)).	• Three persons: John, Paul and Mary
fact(rel (paul, 6, *, john).	• Paul works 6*John
fact(rel(mary, 5, *, paul).	• Mary works 5*Paul
fact(total_value(285)).	• They have to share £285
question(john, paul, mary).	• How much John gets? How much Paul gets? How much Mary gets?

"An equation expresses a part of the condition. The student should be able to tell which part of the condition is expressed by an equation that he brings forward - and which part is not yet expressed." In the TEACHER model the part of the condition that is not expressed is problem specific or "Knowledge of context i" in Appendix 1, or is in part of "General knowledge" in Appendix 1 In our example we have to add a new object to the textbase. The new object is obtained from object(john,paul,mary) by adding operations that bind these objects to a larger object:

Enlarged textbase	**The notes of a student or of a teacher**
operations_between_objects[+,+])	• How much do John Paul and Mary get altogether?

"An equation expresses the same quantity in two different ways. The student should be able to tell which quantity is so expressed." The quantity is, in our examples (Table 4), the sum or difference of ages in age problems, the sum of money in the above mentioned money sharing problems, and the sum of angles in triangle problems. The similar types of quantities can be found in the corresponding problem types (Fig. 9).

"Of course, the student should possess the relevant knowledge without which he could not understand the problem." Computers can not analyse natural language. Therefore this kind of relevant knowledge is beyond the system's boundaries.

"Many of the usual high school problems are "rate problems". Before he is called upon to such a problem, the student should acquire in some form the idea of "rate", proportionality, uniform change." This is also one of the main aims of the study: to present a method by which the student can learn to handle "rates" or to understand the meaning of relations. The TEACHER model concentrates, in this discipline, on representing conditions of the text by relations. To build the textbase is to build an intermediate representation between the verbal representation of a problem and its symbolic representation as an equation. To teach the students to use an explicit intermediate representation has proven to have significant and long-term effects on the student's mathematical word problem-solving abilities [SeBe-91].

The TEACHER model tries to make explicit those factors that are known to be relevant in the *teaching* of word problem-solving [Poly-65]. The choice of an inappropriate variable is a cause of many errors and the student may even give up trying to solve the problem if he doesn't find the adequate variable. The questions related to the choice of a variable are discussed in the next section. After choosing the variable the we are able to construct a context sensitive model of the problem (Appendix 1).

The context sensitive model contains all the information, derived from text and from extra knowledge that is needed to solve the problem. If a student can form a context sensitive model, he knows all the relevant features of a problem. He knows exactly what extra knowledge is needed and he knows what are the unknowns in the problem. Often a student bypasses the context sensitive model and goes directly to the context free model. The context free model has all the information needed to form the equation, but the context free model does not include any information about which objects are to be solved. In other words, it is quite usual that students solve problems by building equations, but they do not know what the answer represents in the problem text. If students want to write the answer correctly, they have to go back to the text and do the analysis process once more. If, however, they have formed a context sensitive model, abstractly or formally, they have at the same time carried along the meaning of the variables.

The benefit of forcing the students to build a context sensitive model is that it is then impossible to rush towards the solution: the student has to think what he or she is really doing. A teacher might encourage the students to make the following notes when he or she teaches how to solve word problems:

- John = x
- Paul = 6*John
- Mary = 5*6*John
- John + Paul + Mary *is equal to* the total pay to be shared.
- total pay = £285

The students are not normally encouraged to make context sensitive models. Rather they are instructed to form a context free model and then to go back to the text and check once more what the real question was. This easily leads to the isolation of the semantic part of the word problem from the algebraic part of the word problem. This might be one reason why so many students solve mathematical word problems without really understanding what they are doing [Reuss-88]. The TEACHER model presents one possible way that can be used to avoid some of the usual mistakes made in mathematical word problem-solving. Thus the model may be of some practical value in delivering instruction on mathematical word problem-solving. This may be done through ITS version of the TEACHER or through the traditional algebra curriculum.

5.3. THE TEACHER AS A TEST BENCH OF PROBLEM-SOLVING PROCESSES

The logic based model of mathematical word problems highlights some of the features of problem solving. The model also raises some new questions. Some old questions of mathematical problem solving can also be tested with the prototype. Before considering these questions we look at what restrictions the model has.

The TEACHER system has been designed in a restricted area of the curriculum of mathematics [Anon-82]. The Finnish seventh graders are supposed to use only about 15 lessons to learn the basic concepts of equation. Six lessons, or two weeks, are used in mathematical word problems. Our selection of a well defined area has the following theoretical advantages. First, it is possible to thoroughly manage a condensed area. Second, new theories may be developed and tested. New theories can be tested afterwards in more comprehensive settings. In addition

to these theoretical features, there is also a pragmatical reason: it is impossible to develop the first version of a system like TEACHER in a complicated area.

In the following we shall discuss some of the theoretical problems that may be studied through the TEACHER system. Some of these questions have been discussed in the previous sections. These two are related to the overall problem-solving process and to the finding of an appropriate representation intermediate between an equation and the text.

TEACHER system can be used to answer the following questions: is it possible to know when extra knowledge is needed and when extra knowledge is not needed to solve a problem i.e. when the textual representation of a word problem contains all information needed to solve the problem? The question may be unsolvable in general settings, but it is interesting to find out if we are able to give an exact answer to the question in a restricted area. If this is possible, then new and unexplored questions arise. The first one is: are we able to enlarge the set of problems and what kind of problems are outside these well-defined problems?

In order to be able to solve the above mentioned problems, we have to analyse thoroughly the transformation between the word problem and the textbase. Basically, the textbase consists of objects (persons, angles,...) , attributes of these objects (age, pay, angle), relations between the objects (three times older than, 60° greater than...), operations between the objects (A+B+C) and possibly the values of some of the objects (total_pay is 360 mk...). We have to consider those features that we have not been forced to tackle in our simplified examples. Doing this detailed analysis of the knowledge representation allows us also to get a more thorough understanding of the different problem domains.

If a student knows the essential features of a word problem (i.e. the student is able to form the textbase) and knows what kind of extra knowledge is needed (i.e. the student is able to form an enlarged textbase), he or she still has a major sub-problem to solve: how to select the right variable? If a wrong variable is selected, the problem may become more complex or even unsolvable for a seventh grader.

Before we can answer the above mentioned questions we have to answer a question such as: what are the steps that are related to the choice of the variable and how many steps are there? If we are able to solve this question, we at the same time have a logic program that can select an appropriate variable. But there may be many different ways to attain the right variable. A variable may be selected

according to some technical features: we can select the variable according to the position of the object in the text (e.g. we may select the first possible variable) or we may simply select the object of comparison. These features may be implemented in the system so that the system is able to predict the variable. It is still an open question as to how many steps there are and what are the actual steps of the selection process.

Some psychological features of the text may also have an influence on the student's ability to solve a problem i.e. word problems that are related to areas which students like, dislike, are interested in, that are from unknown areas etc. It is possible to test these features through the TEACHER system because the underlying mathematical structure of a word problem may be kept invariable and change only the textual form of the word problem.

6. SUMMARY

In this study a logic based model for mathematical word problem-solving was developed. The model advances through several abstraction levels from a word problem to an equation. The transformations between the abstraction levels are relational. The model emphasises relational features of mathematical word problem-solving.

The properties of a logic programming language, Prolog, as a modelling language for mathematical word problem solving are analysed and applied. Prolog's inference mechanism is based on resolution. We point out that resolution is related to mathematical analysis. This study shows that it is possible to model a problem solving process and simultaneously get an executable Prolog-program of that model.

The curricula in mathematics are changing in many countries. One explanation for the change is the use of computers. This study shows that the use of graphical calculators will force further change in these new curricula in the near future. The emphasis will then be on topics in which man is superior to a computer e.g. as word problems.

The study suggests a new method for analysing and categorising the textual forms of mathematical word problems. It is possible to write the grammars of different word problem types using DCG notation. This method may be applied to textbook analysis to answer question such as: How many different categories there are in some subset of a curriculum? Do textbooks have different amount of categories? Are there great differences of the grammars within the same problem domain? How do these features of the teaching material affect to students overall performance to solve word problems?

As a result of the modelling, we have constructed the inner structure of mathematical word problems. This relational structure can be used as a framework

for empirical studies of word problem-solving. By keeping the syntactical part invariable, it is possible to investigate questions such as: do semantical factors of word problems have an affect on the students' overall performance in problem-solving?

The TEACHER prototype as such as it is now is only a minor prototype and it should be seen as a framework for building larger programs. Our main purpose was to build a model of word problem solving and not to build an ITS. Much work is needed to develop the TEACHER to a robust program that could be used to support education. We have described the TEACHER prototype at a declarative level. The procedural details are important in large and relational Prolog programs. In addition the inner levels of the model, the enlarged textbase and the context sensitive level, are problem type specific i. e. they are not implemented in general terms. A further research question is: is it possible to implement all the levels using DCGs? Or in general: is it possible to implement the different levels so that new problem types can easily be added to the system? These questions would probably need both a thorough textbook analysis and experiments in program construction. In any case, the bi-directional model together with its ability to generate word problems could be used as an essential part of a new type of method for teaching how to solve mathematical word problems.

Taken together, the results of the present work indicate that mathematical word problems found in standard textbooks have a logical structure that is inherited from the equation, i.e. word problems are isomorphic with the underlying equation. The analysis of word problems can thus start from the structure of the equation. Logic programming has proven to be a useful tool in this modelling process. The modelling advances through logically connected abstraction levels towards the textual form of the word problem. As a result of this analysis we also get a model from word problems to equations. This model quite closely follows Descartes' scheme as presented by Polya.

An answer to our original research question is that to improve the teaching of word problems we should introduce Polya's methods of teaching word problem-solving. Special emphasis should be put on the introduction of the relational features of word problems. This study also raises the following question: do the present semantically poor textbook problems teach only a procedural way of solving word problems?

REFERENCES

[Ande-81] J.R.Anderson (ed.), Cognitive Skills and Their Acquisition. Hillsdale, US: Erbaum, (1981).

[AnBo-90] J.R.Andersson, C.F.Boyle, A.T.Corbett and M.W.Lewis, Cognitive Modelling and Intelligent Tutoring, Artificial Intelligence, vol. 42, No 1, February, 7-49, (1990).

[Anon-82] Peruskoulun matematiikan oppimäärä ja oppimääräsuunnitelma (The Course and Syllabus of Comprehensive School Mathematics), kouluhallitus, Valtion painatuskeskus, (1982) (in Finnish).

[Anon-89] Kognitiotieteen tutkimusohjelma (The Research Programme of Cognitive Science), Suomen Akatemia, kognitio-tieteen jaosto, (1989) (in Finnish).

[Anon-93] International Baccalaureate, Group 5 Mathematics Guide, International Baccalaureate Organisation, Geneva, Switzerland, (1993).

[BaFe-81] A.Barr and E.A.Feigenbaum, The Handbook of Artificial Intelligence, vol. I, Pitman, (1981).

[Bobr-68] D.G.Bobrow, Natural Language Input for a Computer Problem-Solving System, in M.Minsky (ed.), Semantic Information Processing, The MIT Press, (1968).

[Boch-76] G.V.Bochmann, Semantic Evaluation from Left to Right, Communications of the ACM 19,2,55-62, (1976).

[Bode-90] M.Boden (ed.), The Philosophy of Artificial Intelligence, Oxford University Press, (1990).

[Brat-86] I.Bratko, Prolog Programming for Artificial Intelligence, Addison-Wesley , Wokingham (1986).

[BrLa-84] D.L Briars and J.H. Larkin, An Integrated Model of Skill in Solving Elementary Word Problems. Cognition and Instruction, Vol. 1, No 3, p. 245-296, (1984).

[Chom-57] N.Chomsky, Syntactic Structures, Mouton, The Hague, (1957).

[Chom-72] N.Chomsky, Language and Mind, Enlarged Edition, H.B.Janowich Inc. New York, (1972).

[Clan-86] W.J.Clancey, From GUIDON to NEOMYCIN and HERACLES, AI Magazine August, 40-77, (1986).

[ClSo-90] W.J.Clancey and E. Soloway, Artificial Intelligence and Learning Environments: Preface, Artificial intelligence, vol. 42, No 1, February, 1-6, (1990).

[ClMc-88] K.L.Clark, F.G.McCabe, N.Johns and C.Spencer, LPA MacPROLOG Reference Manual, Logic programming Associates Ltd, London, (1988).

[ClMe-84] W.Clocksin and C.Mellish, Programming in Prolog, Springer-Verlag, Heidelberg, (1984).

[CoSm-88] A.Collins and E.E.Smith, Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence. Morgan Kaufman, California, (1988).

[Colm-78] A.Colmerauer, Metamorphosis Grammars, in L.Bolc(ed.) Natural Language Communication with Computers, Lecture Notes in Computer Science, 133-189 Springer-Verlag, Berlin, (1978).

[CoVe-85a] E.DeCorte and L.Verschaffel, Beginning First Grader's Initial Representation of Arithmetic Word Problems. Journal of Mathematical Behaviour,No 4, 3-21, (1985).

[CoVe-85b] E.DeCorte and L.Verschaffel, Influence of Rewording Verbal Problems on Children's Problem Representations and Solutions, Journal of Educational Psychology, Vol. 77, No 4, 460-470, (1985)

[CoVe-86] E.DeCorte and L.Verschaffel, Eye-Movement Data as Access to Solution Process of Elementary Addition and Subtraction Problems, Paper presented in a symposium on "International research on children's solutions of arithmetic word problems", held at the Annual meeting of the AERA, April, (1986).

[CuFa-81] K.Culik and M.Farah, Linked Forest Manipulation Systems- A Tool for Computational Semantics, in J.T.Tou (ed.) Advances in Information Systems Science Volume 8, Plenum Press, (1981).

[Drey-90] H.L.Dreyfus, Is Socrates to Blame for Cognitivism? in B.Görantzon and M.Florin (eds.), Artificial Intelligence, Culture and Language: On Education and Work, Springer-Verlag, London, (1990).

[DrDR-86] H.Dreyfus and S. Dreyfus, Mind over Machine, The Free Press, (1986).

[DuMa-87] C.Dudley-Marling and R.Owston, The State of Educational Software: A Criterion-Based Evaluation. Educational Technology, March 87, 25-29, (1987).

[Duff-90] J.A.Duffield, Designing Computer Software for Problem-Solving Instruction, Educational Technology Research & Technology, Vol. 39, No 1, 50-62, (1990).

[EbBr-88] R.E.Eberts and J.F.Brock, Computer-Based Instruction in M. Helander (ed.), Handbook of Human-Computer Interaction, Elsevier Science Publishers B.V., North-Holland, (1988).

[Egge-90] A.E.Eggert, A Rebuttal to "A Role for AI in Education: Using Technology to Reshape Education". Journal of Artificial Intelligence in Education, Vol. 1(3) Spring, 3-9, (1990).

[ElSc-90] R.Elio and P.B.Scharf, Modelling Novice-to-Expert Shifts in Problem-Solving Strategy and Knowledge Organization, Cognitive Science No 14, 579-639, (1990).

[Elor-74] K.T.Eloranta, Heuristiikat ja heuristisuus: Käsittelyongelmista ja niiden ratkaisemisen metodologiasta hallinto-opin näkökulmasta (Heuristics and the Heuristic Procedure: on Problem Solving and its Solving Methodologies from the Viewpoint of Administration Science), Tampereen yliopisto, Kirjapaino Hermes Oy, Tampere, (1974) (in Finnish).

[Elor-79] K.T.Eloranta, Menetelmäeksperttiyden analyysi menetelmäkoulutuksen suunnittelun perustana (The Analysis of Methodological Competence as the Base of Planning Methodological Training), Tampereen yliopisto, Tampereen yliopiston keskusmonistamo, (1979) (in Finnish).

[Elst-79] Elstein, Schuman and Sprafka, Medical Problem-solving, Harvard University Press, London, (1979).

[GeNi-88] M.R.Genesereth and N.J.Nilsson, Logical Foundations of Artificial Intelligence, Morgan Kaufmann Publishers Inc.,California, (1988).

[Gree-87] J.G.Greeno, Instructional Representations Based on Research about Understanding, in A.H.Schoenfeld (ed.), Cognitive Science and Mathematics Education, Lawrence Erlaub Associates, Publishers, Hillsdale New Jersey, 61-88, (1987).

[GyGu-89] M.Gyssens, J.Paredaens and D. Van Gucht, A Grammar-based Approach towards Unifying Hierarchical Data Models (extended abstract), ACM SIGMOND, Oregon, (1989).

[HaKi-89] R.Hall, D.Kibler, E.Wenger, and C.Truxaw, Exploring the Episodic Structure of Algebra Story Problem-solving. Cognition and Instruction No 6, 223-283, (1989).

[Haut-88] A.Hautamäki(ed.), Kognitiotiede, Gaudeamus, (1988) (in Finnish).

[HeKu-88] M.Heinonen, A.Kupiainen and E.Sainio, Plussa 7: matematiikka (Plus 7: Mathematics), Otava, Keuruu, (1988) (in Finnish).

[Hein-89] S.Heinämaa, Ajattelun kielioppi (The Grammar of Cognition) in S.Heinämaa and J.Tuomi, Ajatuksia synnyttävät koneet (The Machines that Create Thoughts), WSOY, Helsinki, 81-100, (1989) (in Finnish).

[Hiet-89] P.Hietala, Applications of AI in Education - an overview, in H.Jaakkola and S.Linnainmaa (eds.) Proc. of the Second Scandinavian Conference on Artificial Intelligence SCAI ´89, Tampere, Tampere University of Technology, Finland, (1989).

[Hoik-90] T.Hoikkala, Teun A. van Dijkin diskurssianalyysi (Teun A. van Dijk's Discourse Analysis), in Klaus Mäkelä (ed.) Kvalitatiivisen aineiston analyysi ja tulkinta (The Analysis and Interpretation of Qualitative Corpus), Gaudeamus, Helsinki, (1990) (in Finnish).

[HoUl-79] J.E.Hofcroft and J.D.Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, (1979).

[Järv-93] P. Järvinen, Notes on Assumptions of User Modelling, University of Tampere, Department of Computer Science, Series of Publications A, A-1993-2, March, (1993).

[Kahn-89] M.G.Kahn, Model based Interpretation of Time-Ordered Medical Data. Dissertation, University of California San Francisco, (1989).

[Kilp-87] J.Kilpatrick, Problem Formulating: Where Do Good Problems Come from?, in A.H.Schoenfeld (ed.), Cognitive Science and Mathematics Education, Lawrence Erlaub Associates, Publishers, Hillsdale New Jersey, (1987).

[KiGr-85] W.Kintsch and J.G.Greeno, Understanding and Solving Word Arithmetic Problems. Psychological Review, 92, No 1, 109-129.

[KiDi-78] W.Kintsch and T.A. van Dijk, Toward a Model of Text Comprehension and Production. Psychological Review, 85, 363-394, (1978).

[KlSz-85] F.Kluzniak and S.Szpakowicz, Prolog for Programmers, Academic Press, London, (1985).

[Kowa-79] R.Kowalski, Logic for Problem-solving, Elsevier North Holland, New York, (1979).

[Kowa-87] R.Kowalski, Directions for Logic Programming in W.Brauer and W.Wahlster (eds.), Wissenbasierte Systeme, 2. Internationaler GI-Gongreß, München, Springer Verlag, (1987).

[Lars-88] S.Larsen, New Technologies in Education: Social and Psychological Aspects, in F.Lovis and E.D.Tagg (eds.), Computers in Education, Elsevier Science Pub.,Amsterdam, (1988).

[Lehn-88] H.Lehning, Mathematics of Tomorrow, in F.Lovis and E.D.Tagg (eds.), Computers in Education, Elsevier Science Pub., Amsterdam (1988).

[LeVa-91] E.Lehtinen, M.Vauras and A.Alaja (ed.), The Fourth European Conference for Research on Learning and Instruction, abstract volume, Turku, Dept. of Education, Univ. of Turku, (1991)

[Lewi-81] C.Lewis, Skill in Algebra in J.R.Anderson (ed.), Cognitive Skills and Their Acquisition. Hillsdale, US: Erbaum, (1981)

[Mali-92] P. Malinen, Looginen ajattelu matematiikan opetuksessa (Logical Thinking in the Teaching of Mathematics), University of Jyväskylä, Departmet of Teacher Education, Research 49, Jyväskylä, (1992) (in Finnish).

[Marj-90] E.Marjomaa, Intentionaalisuus ja tekoäly (Intentionalism and Artificial Intelligence), Filosofisia tutkimuksia Tampereen yliopistosta, Vol. XI. Tampereen yliopiston jäljennepalvelu, (1990) (in Finnish).

[Matz-82] M.Matz, Towards a process model for high-school algebra errors in D.Sleeman and J.S.Brown (eds.), Intelligent Tutoring Systems,. 25-50, Academic Press, London (1982).

[McHa-69] J.McCarthy andP.J.Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence, in Meltzer and Michie (eds.), Machine Intelligence 4, New York, p. 463-502, (1969).

[McCo-87] M.McCord, Natural Language Processing in Prolog, in A.Walker (ed.), Knowledge Systems and Prolog, Addison-Wesley, 291-324, (1987).

[McDe-87] D.McDermott, A Critique of Pure Reason, in M.Boden (ed.), The Philosophy of Artificial Intelligence, Oxford University Press, 1990, 206-230, (1987).

[Meis-90] V.Meisalo, Tietotekniikan opetuskäytön uusia virtauksia (New Ideas in Using Computer Technology in Schools), in V.Meisalo and K.Sarmavuori (ed.), Ainedidaktiikan tutkimus ja tulevaisuus III (Subject Didactical Research and Future III), Yliopistopaino, Helsinki 1990 (in Finnish).

[MiCa-83] Michalski, Carbonell and Mitchell (eds.), Machine Learning: An Artificial Intelligence Approach, Morgan Kauffmann, (1983).

[MiCa-86] Michalski, Carbonell and Mitchell (eds.), Machine Learning: An Artificial Intelligence Approach, Volume II, Morgan Kauffmann, (1986).

[MiBa-89] D.Michie and Michael Bain, Machines that Learn and Machines that Teach, in H.Jaakkola and S.Linnainmaa (eds.) Proc. of the Second Scandinavian Conference on Artificial Intelligence SCAI ´89, Tampere Finland, Tampere University of Technology, 2-27, (1989).

[MiPa-89] D.Michie, A.Paterson and J. Hayes-Michie, Learning by Teaching, in H.Jaakkola and S.Linnainmaa (eds.) Proc. of the Second Scandinavian Conference on Artificial Intelligence SCAI ´89, Tampere Finland, Tampere University of Technology, 413-437, (1989).

[Miet-87] R.Miettinen, Kognitiivisen oppimisnäkemyksen tausta (The Background of the Cognitive School), Valtion painatuskeskus, (1987) (in Finnish).

[Mins-68] M.Minsky (ed.), Semantic Information Processing, The MIT Press, (1968).

[NaKi-92] M.J.Nathan, W.Kintsch and E.Young, A Theory of Algebra-Word-Problem Comprehension and Its Implications for the Design of Learning Environments. Cognition and Instruction, Vol. 9, No 4, p. 329-389, (1992).

[Neiss-82] U.Neisser, Kognitio ja todellisuus (Cognition and Reality), Weilin & Göös, Espoo, (1982) (in Finnish).

[NeSi-63] A.Newell and H.Simon, GPS: A program that simulates Human Thought, in E.A.Feigenbaum and J.Feldman (eds.), Computers and Thought, R. Oldenbourg KG., 1963 , Reprinted in Readings in Cognitive Science, Morgan Kaufman, California, 453-460, (1988).

[NeSi-76] A.Newell and H.Simon, Computer Science as Empirical Enquiry: Symbols and Search, in M.Boden (ed.), The Philosophy of Artificial Intelligence, Oxford University Press, 1990, 105-132, (1976).

[Niin-83] I.Niiniluoto, Tieteellinen päättely ja selittäminen (Scientific Argumentation and Explanation), Otava, (1983) (in Finnish).

[Niin-89] I.Niiniluoto, Informaatio, tieto ja yhteiskunta - Filosofinen käsiteanalyysi (Information, Knowledge and Society - a Philosophical Concept Analysis), Valtion painatuskeskus, (1989) (in Finnish).

[Nwan-90] H.S.Nwana, Intelligent Tutoring Systems: an Overview. Artificial Intelligence Review, No 4, 251-277, (1990).

[PaVo-89] J.Paasonen, E.Voutilainen and H.Kalla, Ahaa 7: matematiikkaa (Ahaa 7: Mathematics), WSOY Porvoo, (1989) (in Finnish).

[Pape-80] S.Papert, Mindstorms: Children, Computers and Powerful Ideas. Basic Books, New York, (1980).

[PaSq-90] S.J.Payne and H.R.Squibb, Algebra Mal-Rules and Cognitive Accounts of Error. Cognitive Science No 14, 445-481, (1990).

[Pea-87] R.D.Pea, Cognitive technologies for Mathematics Education, in A.H.Schoenfeld (ed.), Cognitive Science and Mathematics Education, Lawrence Erlaub Associates, Publishers, Hillsdale New Jersey, (1987).

[PeWa-80] F.C.N.Pereira and D.H.D.Warren, Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence No 13, 231-278, (1980).

[Pew-88] R.W.Pew, Human Factors Issues in Expert Systems in M. Helander (ed.), Handbook of Human-Computer Interaction, Elsevier Science Publishers B.V. (North-Holland), (1988).

[Piet-86] J.Pietarinen, Tietokone kaltaiseni? (A Computer, Is it like me?), Tiede 2000, No 1, (1986) (in Finnish).

[Poly-45] G.Polya, How to Solve It, A New Aspect of Mathematical Method, first published by Princeton University Press 1945, second edition, Penguin Books, Great Britain, (1990).

[Poly-65] G.Polya, Mathematical Discovery: On Understanding, Learning, and Teaching Problem-solving, New York, John Wiley & Son, (1965).

[PsMa-88] J.Psotka, L.D.Massey and S.A.Mutter, Intelligent Tutoring Systems: Lessons Learned. Lawrence Erlaub Associates, Publishers, Hillsdale New Jersey, (1988).

[Resn-89] L.B.Resnich(ed.), Knowing, Learning and Instruction: Essays in Honour of Robert Glaser. Lawrence Erlaub Associates, Publishers, Hillsdale New Jersey, (1989).

[Reus-87a] K.Reusser, SituationProblemSolver SPS: Beschreibung der Implementation in INTERLISP-D, Universität Bern, Abteilung Pädagogische Psychologie, Bern, Switzerland, (1987).

[Reus-87b] K.Reusser, The Computer's Effort after Meaning: Word Problem-solving as a Process of Text Comprehension and Mathematization, Universität Bern, Abteilung Pädagogische Psychologie, Bern, Switzerland, (1987).

[Reus-88] K.Reusser, Problem-solving Beyond the Logic of Things: Contextual Effects on Understanding and Solving Word Problems, Instructional Science, Vol. 17 No 14, 309-338, (1988).

[Reus-90] K.Reusser, From text to Situation to Equation: Cognitive simulation of Understanding and Solving Mathematical Word Problems, in H.Mandl, E. De Corte, N.Bennett and H.F.Friedrich (Eds.) Learning and Instruction. Pergamon Press, Oxford, 477-498, (1990).

[ReKä-90] K.Reusser, A.Kämpfer, M.Sprenger, F.Staub, R.Stebler, and R.Stüssi, Tutoring Mathematical Word Problems Using Solution Trees. Research report No 8 University of Bern, Switzerland April, 477-498, (1990).

[ScEd-89] R.C.Schank and D.J.Edelson, A Role for AI in Education: Using Technology to Reshape Education. Journal of Artificial Intelligence in Education, Vol. 1(3) Winter 1989/1990, 3-20, (1989).

[Scho-87] A.H.Schoenfeld (ed.), Cognitive Science and Mathematics Education, Lawrence Erlaub Associates, Publishers, Hillsdale New Jersey, (1987).

[Sear-90] J.R.Searle, Cognitive Science and the Computer Metaphor in B.Göranzon and M.Florin (eds.), Artificial Intelligence, Culture and Language: On Education and Work, Springer-Verlag: London, (1990).

[Sear-92] J.R.Searle, The Rediscovery of Mind, The MIT Press, London, (1992).

[Sein-74] K.Seinelä, Lukion mekaniikan ohjelmoituja tehtäviä (Upper Secondary Shool Programmed Mechanical Assignments), Otava, Helsinki, (1974) (in Finnish).

[SeBe-91] D.H.Sellke, M.J.Behr, Using Data Tables to Represent and Solve Multiplicative Story Problems. Journal for Research in Mathematics Education, Vol. 22, No 1, p 30-38, (1991).

[Shap-87] S.C.Shapiro(ed.), Encyclopaedia of Artificial Intelligence, J.Wiley & Sons, New York, (1987).

[Shoh-86] Y. Shoham and D.V.McDermott, Knowledge Inversion, in AAAI National Conference On Artificial Intelligence, Texas August 6-10, (1986).

[Silv-86] B.Silver, Precondition Analysis: Learning Control Information in R.S.Michalski, J.G.Carbonell and T.M.Mitchell, Machine Learning: An Artificial Intelligence Approach, vol. II. Morgan Kaufmann Pub. Inc., Los Altos California, (1986).

[Silv-87] E.A.Silver, Foundations of Cognitive Theory and Research for Mathematics Problem-solving Instruction, in A.H.Schoenfeld (ed.), Cognitive Science and Mathematics Education, Lawrence Erbaum Associates, Publishers, Hillsdale New Jersey, (1987).

[Simo-83] H.A.Simon, Why Should Machines Learn, in Michalski, Carbonell and Mitchell (eds.), Machine Learning: An Artificial Intelligence Approach, Morgan Kauffmann, (1983).

[SlBr-82] D.Sleeman and J.S.Brown (eds.), Intelligent Tutoring Systems, Academic Press, London, (1982).

[Sowa-84] J.F.Sowa, Conceptual Structures, Information Processing in Mind and Machine, Addison-Wesley, (1984).

[Stee-90] Luc Steels, Components of Expertise, AI Magazine, vol. 11, No 2, Summer, 29-49(1990).

[StSl-91] R.M.Steinman, D.H.Sleeman and D.Ktorza, Algebra students' knowledge of Equivalence of Equations, Journal for Research in Mathematics Education, Vol. 22, No 2, 112-121, (1991).

[StSh-86] L.Sterling and E.Shapiro, The Art of Prolog, The MIT Press, London, (1986).

[Szpa-87] S.Szpakowicz, Logic grammars, BYTE 12, 9, 185-195, (1987).

[Thae-86] J.S.Thaeler, A New Solution to an Old Problem - Solving Word Problems in Algebra. Mathematics Teacher, Vol. 18, No 3, 163-179 Dec., (1986).

[Vout-89] T.Voutilainen, J.Mehtäläinen, I.Niiniluoto, Tiedonkäsitys, Kouluhallitus and Valtion painatuskeskus (The Concept of Knowledge) , Helsinki, (1989) (in Finnish).

[Walk-87] A.Walker, Expert Systems in Prolog, in A.Walker(ed.), Knowledge Systems and Prolog, Addison-Wesley, (1987).

[Weiz-77] J.Weizenbaum, Computer Power and Human Reason: From Judgement to Calculation. W.H.Freeman and Company, San Francisco, (1977).

[Weng-87] E.Wenger, Artificial Intelligence and Tutoring Systems: Computational and Cognitive approaches to the Communication of Knowledge. Los Altos, California: Morgan Kaufmann, (1987).

[WhFr-90] B.Y.White and J.R.Fredriksen, Causal Model Progressions as a Foundation for Intelligent Learning Environments, Artificial Intelligence, vol. 42, No 1, February, (1990).

[Wils-87] W.G.Wilson, Programming Techniques in Prolog, in A.Walker(ed.), Knowledge Systems and Prolog, Addison-Wesley, (1987).

[ZhGr-88] J.Zhang and P.W Grant, An Automated Difference-list Transformation Algorithm for Prolog, European Conference on Artificial Intelligence, Munich, August 1-5, 320-325, (1988).