1998

# A case-based integration tutorial system.

Qin. Xu
*University of Windsor*

# INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

# A Case-Based Integration Tutorial System

by

Qin Xu

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor
Windsor, Ontario, Canada
1998

# ABSTRACT

The *case-based reasoning* (CBR) paradigm models how the reuse of stored experiences (cases) contributes to expertise. A case-based reasoning system solves a new problem by retrieving stored information on previous problem-solving episodes and adapting it to suggest solution(s) for the new problem. The results are then themselves added to the reasoner's memory (if necessary)in the form of new cases for future reuse.

In this thesis, we build an intelligent tutorial system, with the case-based reasoning paradigm, which is intended to help university/college students solve integration problems in *Calculus* course. The system captures a fairly large amount of experience in solving certain types of integration problems from experienced human problem-solvers. The experience is mainly stored in the form of cases. The system simulates the way in which professors and textbook examples help students. For a given integral, the system searches its case base to find integrals similar to the given one, and makes the hints and answers for the retrieved integrals available to students. The system does not provide solutions to integrals, in order to encourage students to carry them out first-handed. The system also learns by retaining new cases into its case base.

The system is built as an internet application, so that it can be accessed through the Internet by anyone anywhere anytime.

*To my Mom, Dad,*

*and*

*to Zhiguo and Kevin*

# ACKNOWLEDGEMENTS

I would like to take this unique opportunity to express my heartfelt gratitude to my supervisor, Professor *Liwu Li*, for his excellent guidance and unremitting encouragement throughout the preparation of this thesis. I indeed count myself fortunate to have worked with someone like him - with very genuine care, interest and support.

I would also like to thank Professor *R. Frost* and Professor *T. Traynor* for their interest in my work, and their invaluable comments and suggestions.

My special thank goes to my wife *Zhiguo Hu*. Without her constant support and encouragement, I would not have achieved what I presented in this thesis.

# Table of Content

# Introduction

*Case-based reasoning* is a relatively recent approach to *problem-solving* and *learning* that has been attracting an increasing amount of attentions in the *AI* community. Over the past a few years, case-based reasoning has grown from a rather specific and isolated research area to a field of widespread interest. At the time being, many computer scientists from all over the world are working on this subject. The activities are rapidly growing - as seen by the increased number of research papers, workshops, availability of commercial produces, and reports on applications in regular use.

As a problem-solving and learning paradigm, case-based reasoning is fundamentally different from other traditional *AI* approaches in many respects. Roughly speaking, case-based reasoning means solving a new problem by remembering a previous similar situation and by reusing information and knowledge of that situation. Instead of relying solely on *general knowledge* (such as *rules*) of a problem domain, case-based reasoning utilizes the *specific knowledge* of previously experienced concrete problem situations, which are called *cases*. For a new problem, a *CBR* system solves the problem by finding a similar past case from its *case-base* and reusing it in the new problem situation. Another important feature of *CBR* is that it is an approach to incremental and sustained learning. A *CBR* system can learn from its own problem-solving experiences by retaining them and making them immediately available for future problems.

*CBR* approach is intuitively plausible because its reasoning process resembles what we human beings do quite often in many professional activities and even in day-to-day

commonsense reasoning. For example, doctors often use their experience with previous patients to help them diagnosis and treat new patients. Lawyers tend to use similar tactics and arguments for similar cases. As another example, we notice that, when people order a meal in a restaurant, they often base their decision on their past experience in that restaurant and/or those like it.

*CBR* has been found very useful and effective in many types of problem domains, such as planning, design and diagnosis. It has been used to perform a variety of reasoning tasks such as situation assessment, trouble-shooting, solution-evaluation and repairation, classification and scheduling, *etc.*. Hundreds of *CBR* systems have been built around the world. Many of them have been put into practical use. There are even commercially available *CBR* tools that help people build their own *CBR* systems.

In this thesis, we attempt to build an intelligent tutorial system using case-based reasoning approach. This system is intended to help university students, in a natural way, in solving certain mathematical problems they may encounter in their first year *Calculus* course. It is also intended to be an internet application, so that it can be accessed by any student anywhere and anytime with ease.

In years of mathematics teaching, I have seen a lot frustration in teaching and learning *integration*, which is perhaps one of the most fundamental concepts in *Calculus* (or even in Mathematics). In the majority of universities and colleges, *Calculus* is a mandatory course for science and engineering students. It is also required for students in business, education and other disciplines in many universities. For most students, the very first challenge in their math training is probably the integration.

The difficulties in solving an integration problem largely come from the fact that there are few effective rules available which students can follow. To be more precise, the rules taught in the text book are far too general for a student to apply on specific problems. When facing a new problem, some students might know *which* rule they should use, but few of them know *how* to apply that rule. In this situation, some students turn to the text book to

find some examples similar to their problem. Some students bring the problem to their professor for hints. Some others simply quit after a few trying.

Although we have such powerful computer algebra systems (*CAS*) as *Maple* and *Mathematica* on the market, which are capable of solving a large variety of mathematical problems. Unfortunately, those systems do not help students solve their homework problems, because they provide students very little more than answers. They do not give students any useful clue on how the problems are solved. From the learning point of view, students will not be able to acquire any knowledge, without trying problems. From the teaching point of view, a correct answer to a problem is of course very important, but an answer without solution steps is absolutely unacceptable to teachers.

So, our solution to this problem is to build an *intelligent agent* which is: 1) capable of providing students with something more than just an answer; 2) available to users anywhere and anytime. Observing the way how students get help from examples and how professors help students, we found that past experience in solving integration problems plays a key role in this teaching and learning process. So, *CBR* is a suitable model for building such a system.

The system captures a fairly large amount of experience in solving certain types of integration problems in the form of cases. It simulates the way in which professors and textbook examples help students. When an integration problem is given, the system searches its case base to find some cases similar to the given one. Since the system has the knowledge about how those integrals are solved and their answers, the knowledge becomes immediately available to students. It has to be stressed that, our system provides students with hints and answers, but not solutions. Students still have to work out the solutions all by themselves. In this way, the system helps students without sacrificing their own efforts. They eventually learn things first hand.

As mentioned before, our system is desired to be accessible by users anywhere and anytime. Modern internet technology makes our goal possible. In our implementation,

many internet techniques and tools, such as *CGI*, *Java*, *Perl*, have been used to make the system available to users through the Internet. On the other hand, in the software engineering aspect, we have adopted the object-oriented software engineering methodology throughout the system evolution.

Initial testing shows that the system is quite efficient in helping students solve their integration problems. Especially, when the problem presented to the system is in the system's case base, it can be instantly retrieved by the system's search engine and presented to the user. In the case where a new problem is not in our case base, the system can also find close matches whose methods can lead to the solution of the new problem. The system is also equipped with necessary utilities for retaining new cases so that the system can learn new problem-solving experiences. This learning ability makes the system more and more powerful down the road.

We believe that, when necessary, our system can be scaled up to cover a wider range of mathematical problems. But it will take far more efforts than one person can afford. By building such a relatively small-scale system, we demonstrate that *CBR* can be a proper model for expert systems in the area of mathematical education. On the other hand, it is not our intention to underestimate or substitute the powerful computer algebraic systems such as *Maple* and *Mathematica*. Instead, we try to complement the functionality of the *CAS*s with our system in the mathematical teaching and learning aspect.

In the rest of this thesis, we give a full picture of the problem domain, the methodology, the design and implementation of the system. In *Chapter One*, we discuss the problem domain in great detail and observe how human beings tend to solve the problems in this domain. *Chapter Two* is a general discussion of the methodology which we adopted in building our system, *i.e.*, the case-based reasoning. Topics discussed there include: what is *CBR*, what is a *case*, and how a *CBR* system conducts its reasoning. We also discuss why *CBR* is *the* proper model for the problem we are tackling.

*Chapter Three* reveals some of the system design and implementation details. We

discuss various components of the system, the case structure, and the search and match algorithm. *Chapter Four* is a concluding remark, where we will discuss the performance, problems, and the implications of the system.

# Chapter One

# The Problem and Proposed Solution

As we mentioned in the introduction, the main objective of the project is to find a practical, effective and natural way to help university/college students get through the difficulties of solving integration problems. In this chapter, we conduct an extensive discussion on various aspects of teaching and learning integration. We observe and analyze how integration is taught in textbooks and in classrooms, what makes it so difficult for the students, and how professors and textbooks help students solve their problems. In the last section, we propose a solution to this problem and discuss its feasibility.

## §1.1 Differentiation and integration

*Integration* (also called *antidifferentiation*), together with *differentiation*, are two most fundamental concepts in *Calculus*, perhaps even in all of mathematics. They are also extremely important in many other areas of science and technology, such as physics, chemistry, engineering, *etc.*. In those areas, real-world problems are abstracted into mathematical models. Solutions of the problems eventually rely on the solution of the mathematical models. Most of the mathematical models are in the form of mathematical equations, such as differential equations, integral equations, and linear equations. Solutions

to the equations are eventually reduced to solving certain kinds of integration problems.

Integration and differentiation are inverse operations to each other. Differentiation means finding the *derivative* of a given function. For example, given function $f(x) = sin^2(x)$, its derivative is

$$f'(x) = 2\,sin(x)\,cos(x).$$

The derivative is usually derived from the original function using definition, differentiation rules and derivatives of some basic functions. We will not go into details about what is derivative and how to find derivatives, because it is beyond the scope of this thesis. Integration means finding a function (called *integral*) whose derivative is a given function. For example, an integral of $g(x) = 2\,sin(x)\,cos(x)$ is

$$\int g(x)\,dx = sin^2(x),$$

because the derivative of the right hand side is the original function $g$. This kind of integral is usually called *indefinite integral*. There is another type of integral called the *definite integral*, which is a scalar rather than a function. The definite integral and indefinite integral are independently defined, in different ways. But they are closely related to each other, at least for all the continuous functions (we must point out that most of the functions dealt with in a *Calculus* course are continuous). The relationship between these two types of integrals is clearly described by the *Fundamental Theorem of Calculus* (which can be found in any *Calculus* textbook). Although there are special techniques to find definite integrals, in the majority of situations, we evaluate definite integrals through finding the corresponding indefinite integrals. So, in this project, we focus on the indefinite integral. In the remainder of this thesis, integral always refer to indefinite integral.

Students usually have little trouble in learning differentiation. But many of them do have difficulty in learning integration techniques. For both differentiation and integration, there are some rules that can be used to perform symbolic calculations. In finding the derivative of a particular function, it is usually obvious which differentiation rule one should apply and how to apply it. But it may not be so obvious which technique should be used to integrate a given function. Generally speaking, there is nc rules that absolutely guarantee obtaining an integral of a function. We discuss this aspect in more detail in the following sections.

## §1.2 What makes integration more difficult

Now, let us take a closer look at how differentiation and integration are usually taught in textbooks and in classrooms, and why one is more difficult to learn than the other.

In a standard *Calculus* textbook, differentiation usually starts with the definition, followed by introducing the derivatives of a handful of so-called basic functions, such as $sin(x)$, $cos(x)$, $tan(x)$, $cot(x)$, $ln(x)$, $e^x$, $x^n$, *etc.*. Derivatives of basic functions are derived directly from the definition of derivative and other mathematical techniques. These derivation procedures are relatively difficult for first year students, so they are usually not required to know how to derive these basic functions, but rather to remember the results. These derivatives are frequently used to derive more complicated functions.

The majority of mathematical functions are the combination and/or composition of basic functions. It is difficult and unrealistic to derive every function using the definition. Instead, non-basic functions are usually derived by using the derivatives of basic functions, in other words, they are derived piece by piece. There are a number of rules which govern how to derive a function through deriving its components. They include: the *addition rule*,

*product rule*, *quotient rule*, and *chain rule*. These rules are taught in nearly every textbook. The application of these rules is quite straightforward, because which rule should be applied to a particular function and how it should be applied can be easily determined by the appearance of the function. By carefully following those rules and carrying out necessary calculations, a correct result is almost guaranteed.

The concept of integration is introduced in different ways in different textbooks. Some books define the integral directly as antiderivative, while others define it through the definition of definite integral and the *Fundamental Theorem of Calculus*. No matter how it is defined, every textbook eventually comes to the common point stating that integral and antiderivative are indeed the same thing.

Despite the different approaches in defining the integral, most textbooks teach integration techniques in a similar fashion. Just as in teaching differentiation, integrals of basic functions are introduced first. (The difference here is that these integrals are not calculated through the definition, but through the observation of the derivatives of certain functions). The strategy for integrating non-basic functions is the same as that for differentiation, *i.e.*, integrating them piece by piece and making use of integrals of basic functions. There are also several rules available for this purpose. They include: the *addition rule*, *substitution rule*, and *the rule for integration by parts*. Each of these rules is derived from one of the differentiation rules. For example, the addition rule corresponds to the addition rule for differentiation, the substitution rule corresponds to the chain rule, and the rule for integration by parts corresponds to the product rule.

Although integration rules are similar to those for differentiation, the use of these rules is not as easy and straightforward. As we have mentioned several times, most students find integration a much harder process than differentiation is. My own experience in teaching *Calculus* is that many students do have the ability to determine which rule to apply for a particular problem. Their choices may come from the textbook context (say, what is being

discussed in the section where the problem is from) or from the appearance of the integrand. Generally speaking, it is not very difficult to rule out the applicability of some rules to a particular integrand. There are some commonsense reasoning rules that can be used for this purpose. For example, the addition rule is not proper for an integrand with only one term, the integration by parts is more likely used for an integrand with more than one non-trivial factors (of course, there are exceptions, for example, $ln\,(x)$). Fortunately, the number of integration rules is very small, and if one can rule out one or two of them, then it becomes quite apparent what rule to apply. The real challenge for students is how to apply a chosen rule to a particular integral problem. The difficulty may come from the following aspects.

1) Some of the rules are not specific enough for direct application. Extra information is often needed in order to apply them to a particular integration problem. For example, the substitution rule does not tell what kind of substitution to perform, and the rule for integration by parts does not tell which part of an integrand should be integrated first. Let's look at some concrete examples. For the integral

$$\int \frac{1}{\sqrt{x^2-a^2}}\,dx \quad ,$$

one can easily rule out the applicability of addition rule and integration by parts, so the substitution rule is perhaps the only rule that one could try. But, from the rule itself, one do not get any clue about what kind of substitution to make. Similarly, for the integral

$$\int x^2 sin^2(x)\,dx \quad ,$$

the addition rule is obviously not suitable. Since the integrand is the product of two basic functions of different types, one can hardly find any substitution that would simplify it. Therefore, integration by parts seems to be the best thing to try. But which part should be integrated first?

2) Very often the rules have to be used in conjunction with some other relevant mathematical techniques and/or formulas. Before any rule can be applied, sometimes one has to do something to the integrand, either to simplify it or even to make it a little more awkward (at least in appearance). For example, to evaluate the integral

$$\int sin^2(x)\,dx,$$

one has to apply the formula $sin^2(x) = \frac{1}{2}(1 - cos(2x))$ first before any rule can be used.

As for what kinds of operations have to be performed before integrating a function, one can find absolutely no clue from the rules.

3) For some integrals, multiple rules and steps may be needed, and perhaps mixed with a few other mathematical operations. To make things even worse, sometimes the order in which one applies the rules does matter. Some orders may result in a quick and easy solution, while others may lead to a long journey or even to a dead end. For example, to evaluate integral

$$\int x\,sin(x)\,cos(x)\,dx,$$

both substitution rule and integration by parts are needed. If one starts with applying the formula $\sin(x)cos(x) = \frac{1}{2}sin(2x)$, followed by a simple substitution $y = 2x$, then a single application of integration by parts will finish the job. Otherwise, one may need much more effort to get a correct answer.

In fact, there are many special integration techniques that can be used to evaluate specific types of integrals. But unfortunately, due to the limitation of the textbook volume

and class time, very few of those techniques are covered by textbooks or professors in classes, if any at all.

## §1.3 Problems in teaching and learning integration

In the following, we observe students' behavior when they have trouble with their exercises, analyze what really helps students in the situation, and where the helpful information may come from. We also discuss some other related problems in this teaching and learning process.

When running into trouble with their exercises, students' reaction varies significantly from person to person and from occasion to occasion. Here are some of actions they could possibly take:

1. blindly try anything they could think of,
2. turn to the textbook to find some examples similar to their problem,
3. bring the problem to the instructor, or
4. copy solution from a fellow student or simply give up.

From the learning point of view, blind trial may not be a bad way to learn things. Students can truly learn a lot though their effort. But it may not be time-efficient, especially for a student who's taking five or more courses (which is common for the first year university students). On the other hand, some students may easily get frustrated and give up their effort. There is another shortcoming with this approach. When students get stuck somewhere, it's quite difficult for them to find out what really went wrong. Most of the students do not have good judgement as to whether they applied an improper rule/formula, or they chose the suitable rule/formula but used it in an improper way, or they got

everything right except for a silly careless mistake in their calculation. The frustration may wear out students' patience quickly.

On the other extreme, we believe that very few students would willingly choose the last action unless they run out of options. This is also the situation which professors have been trying very hard to avoid. For example, some professors give students some hints for the problems that they think might be a little hard for students, some others encourage students to discuss and work in groups.

In our observation, actions 2 and 3 are more common and effective approaches taken by students. When students go to a textbook, what they really look for are examples. They usually go through pages trying to find an example which is similar to their problem. If they find such an example, they try to understand how the example problem has been solved and attempt to apply the same method to their own problem. Even though sometimes the method may not directly solve their own problem, students still get hints from the method. For example, someone may try a variant of the method according to the characteristic of their own problem and/or the difference between two problems (say, one may try a similar substitution or a similar mathematical formula, *etc.*). So, a good matching example give students some meaningful ways to try out their own problem and give them a better chance to succeed. On the other hand, when students really understand the example, it becomes their own asset, their own problem-solving experience. They learn more than just how to solve a single problem, they actually acquire a method that might be used to solve a class of problems.

Of course, there are times when they can not find any matching example from their textbook. This is mainly due to the enormous diversity of functions and the limitation of a textbook volume. It also happens quite often that students are unable to identify matching examples in their textbook because of their lack of knowledge and experience. Ideally, if there is a good number of examples available to students in some way and there is a

knowledgeable person who is constantly available to help students in identifying similar examples, students would have a much better chance to solve their problems and hence to learn more.

Professors' role in helping student is quite different from that of textbooks. When students bring their problem to their professor, they expect some hints about how to solve their problem rather than examples. Professors' hints usually helps students more directly, in other words, it can be directly applied to students' problem and almost guarantee a solution. In most of the situations, professors can come up with a hint quickly, without having to try the problem out. So, where do their hints come from? We believe they come from their past experiences in solving the integration problems. They might have met and solved the problem before, or they might have tried a similar problem before. Their knowledge and their past training give them a much better chance to provide a meaningful and reliable hint. Of course, professors' hints are very valuable to students. But there is also a problem with that. Professors are not constantly available. When students need help, professors may not be reachable.

Summing up, from the above observations, we have seen that hints do help students effectively, but helpful hints come from substantial problem-solving experience. Matching examples also help students a great deal in solving their problems, but it takes experience to identify matching examples. So, in this teaching and learning process, experience plays a vital role.

Now, we find that we are in such a contradictory situation: Integration is a relatively hard operation which requires a fairly large number of experience. But students who are supposed to do integration problems do not have much experience. To make things even worse, instructors who do have experience are not always available to students when help is needed.

It is certain that both students and instructors do not want to see students giving up their effort too soon. But, there have to be some convenient and effective ways to help them out of the frustration. For this purpose, some students prepare more than one textbook, and some even hire tutors. These help students quite a lot, but costs them a great deal as well.

## §1.4 Proposed solution

Our proposed solution to this problem is to build an intelligent agent, which is capable of helping students just like textbooks and professors do and is constantly accessible to all students. The system must capture a fairly large number of concrete problem-solving experiences in integration, be capable of retrieving useful experiences when needed, and be available to anyone anywhere and anytime.

The purpose of this system is to give students some help at the time they need it. The system should provide students with hints, rather than solutions, in order to encourage them to carry out the solution by themselves, because we believe that hands-on experience is the best way to learn. Once they accumulate a certain amount of experience, they can eventually get rid of this system and work on problems independently.

Someone may question the necessity of such a system because of the existence of such powerful computer algebraic systems as *Maple* and *Mathematica* in the market. Yes, those systems are capable of solving a large variety of mathematical problems. But from teaching and learning point of view, they do not help students in healthy way because they do not provide any useful information about how the problem can be solved except for an answer. Indeed, to be fair, *Maple* is capable of providing various levels of details on how the system solve a problem (by setting the *infolevel* to different integer values). But those information is hardly useful for students because, for some technical reasons, *Maple* use a number of very complicated algorithms to solve even the simplest integrals. Those techniques are far

too difficult to be carried out by human beings, especially by first year students. So, we believe the proposed system will be a complement to the functionality of those computer algebraic systems.

We would like to make it very clear that, it is not our intention to make this system another "*integration wizard*" that solves hard integration problems. In fact, our system does not actually solve any integration problem, instead it is a sort of knowledge repository that *intelligently* interact with human users. The system captures a large amount of specific problem-solving experiences and knows when and which piece of experience is useful.

In building such an intelligent system, there are three major concerns: 1) what kind of problem-solving schema to adopt, 2) how to acquire the necessary knowledge and 3) what kind of implementation technology to use. To answer the third question, we have to keep in mind that one of the key features of our intended system is its universal accessibility. Modern internet technology provides us various ways to achieve that. We will address numerous implementation problems later in Chapter 3. The knowledge acquisition problem is closely related to the methodology, because different approaches require knowledge in different forms.

There are many different approaches in building an expert system, for example, *knowledge-based* systems, *rule-based* systems, *memory-based* systems, *explanation-based* systems and *model-based* systems. As we mentioned before, the approach we are going to take is the case-based reasoning. We will discuss in detail what *CBR* really is and why we take this approach in the next chapter.

# Chapter Two

# Methodology: Case-Based Reasoning

We are going to build our intelligent agent using the Case-Based Reasoning approach. Before presenting our system, we give a more complete picture of *CBR* methodology in this chapter. The chapter is organized as following. In *Section §2.1*, we present a brief review of the theoretical background of *CBR*. *Section §2.2* discusses some issues regarding the cases, such as case representation, case indexing, and so on. In *Section §2.3*, we discuss how a *CBR* system works and some related technical issues. Finally, we conclude this chapter with an argument on why *CBR* is a suitable approach for our problem domain described in the previous chapter.

## §2.1 The theoretical background of *CBR*

As its name suggested, case-based reasoning uses cases to reason. Therefore, a case-based reasoner's knowledge is largely in the form of cases. Roughly speaking, a case is a description of a concrete problem-solving experience. It usually consists of specific information on various aspects of a particular problem-solving experience, for example, the problem itself (problem description), the index information (where the case fits in the case

base), the solution of the problem, the outcome of the solution (success, failure, remedy needed, *etc.*), and so on. (We will present a more comprehensive discussion on cases in the next section). A *CBR* reasoner usually takes a problem description as its input. It searches its case base to find a best matching case (or cases) and suggests its solution for solving the new problem. Some *CBR* reasoners are even capable of modifying the old solution to fit the new problem. By retaining new cases, many *CBR* reasoners can learn through its own problem-solving experience as well.

The most fascinating feature of the case-based reasoning paradigm is that its reasoning process resembles that of human being. As revealed by some prominent research works, there is a considerable evidence that the retrieval and replication of specific problem-solving experience plays an important role in human problem-solving in a wide range of task contexts. For example, studies support the importance of recall of prior examples in learning a computer text editor [63], learning programming [53], mathematical problem-solving [26, 63], diagnosis by automobile mechanics [45] and physicians [67], explanation of anomalous events [57], and decision-making under time pressure [37, 38].

Ross [64, 65] showed that people learning a new skill often refer back to previous problems to refresh their memories on how to do the task. Research conducted by Kolodner and her group showed that both novice and experienced auto-mechanics use their own experience and those of others to help them generate hypotheses about what is wrong with a vehicle, conduct test for different diagnosis, and finally recognize the real problem [46]. It has also been observed that case recall plays an important role in human reasoning for tasks such as real estate [20] and dispute mediation [68], in which it is difficult to enumerate and weigh all the factors that may be relevant, and in which frequently changing circumstances preclude reasoning from a fixed set of rules.

Klein and Calderwood [37] observed expert decision makers in some complex, dynamically changing situations. In several different natural decision-making situations,

they observed that, experts use analogs to understand situational dynamics, generate options, and predict the effects of implementing an option. Experts also use cases both to suggest solutions that are then adapted and to evaluate solutions and situations. They concluded that, in those situations, use of analogs is far more important than application of abstract principles, rules, or conscious deliberation about alternatives, because analogs of cases provided concrete manifestations of the rules or principles that allowed them to be applied easily. Cases also alert decision makers to causal factors operating during the incident, to anticipate what might happen if a course of action is implemented, to suggest options, and to reassure the decision makers that an option has worked and can be relied upon.

The nature of the case-based reasoning endowses it with many other advantages over traditional problem-solving methodologies as well. *First*, the *CBR* paradigm does not require a total understanding to the problem domain. Our knowledge about the world is so limited that there are many things which we do not understand well, if at all. It is not rare that, in an area, we know how to solve some particular problems without knowing why. Therefore, *CBR* provides us a way of dealing with those areas about which we lack of general knowledge. *Second*, building expert systems might be easier in a case-based paradigm because it is much easier to capture an expert's knowledge in the form of cases than in the form of general rules, especially in those areas which we do not understand well. On the other hand, in order for a *CBR* system to work, its case base need only to cover those types of situations that actually occurred in practice. It is not necessary, sometimes even impossible, to include all possible situations. (But generative systems must account for all problems that are possible in principle [see 47]). *Third*, the case-based reasoning process itself is relatively simple. *CBR* allows a reasoner to solve problems with a minimum effort, because it remembers the past cases, which make it possible to avoid starting from scratch for every problem. *Fourth*, knowledge maintenance for case base is easier than other type of knowledge base. In building any kind of expert system, initial understanding of the problem domain is often imperfect. When new and/or unexpected problems occur,

knowledge re-engineering is necessary. For a *CBR* system, this can often be as easy as adding a few new cases to the case base.

So far, we have solely emphasized the role of specific knowledge in the case-based reasoning paradigm. We do not mean to rule out the role of general knowledge in a *CBR* reasoner. Certain forms of general knowledge are also frequently used in building *CBR* systems. Especially, the searching algorithm and solution adaptation strategy usually consist of certain types of general knowledge. Compared to the role of general rules in other methodologies, its role in a *CBR* system is relatively limited.

The theoretical roots of *CBR* may be traced back to the work of R. Schank on the *theory of dynamic memory* and the central role that a reminding of earlier situations and situation patterns plays in problem-solving and learning [66]. The first system that might be called a case-based reasoner was the system named ***CYRUS*** [1], developed by J. Kolodner [39] at Yale University (Schank's group). ***CYRUS***'s memory structure design was based on Schank's dynamic memory theory and MOP (*Memory Organization Packets*) model. It was basically a question-answering system with the knowledge of various travels and meetings of former US Secretary of State Cyrus Vance.

The main premise of the theory of dynamic memory is that remembering, understanding, experiencing, and learning cannot be separated from each other. The theory suggests that our memories, which are dynamic, change as a result of our experiences due to the new things we encountered, the question that arise on our minds as a result of experiences, and the way we answer these questions. We understand by trying to integrate new things we encounter with what we already know. Thus, understanding causes us to remember old experiences, consciously or unconsciously. Understanding, in turn, allows memory to refine itself, in other words, to be dynamic.

---

[1] Stands for "Computerized Yale Retrieval and Update System"

This *understanding and learning cycle* is the process that drives human reasoning. Knowing where something fits in with what we already know is a prerequisite to reasoning with it. We find it hard to suggest a solution to a problem until we understand the problem. On the other hand, further reasoning about the problem (based on our earlier imperfect understanding of it) may help us to understand it better. We use what we know about a situation to find the closest match in memory, and we use that as the basis for deriving expectations and inferences. As we gather additional information about a situation, we may find closer matches in memory, allowing us to derive better expectations and inferences.

Under this view, finding the right previous experiences (cases) in memory is the key to successful reasoning. The process of identifying the right cases in memory is usually called *reminding*. A frequently asked question is: we have so much knowledge and have so many cases available to us, how do we find the right cases, in other words, how does reminding happen? There are many theories that try to answer this question. The premise of the theory of dynamic memory on this aspect is that the reminding process is closely related to the organization of memory structures. It suggests that the human memory structures are *labeled* (indexed) in a way that make them easily accessible when needed. One of the main memory structure proposed in the theory of dynamic memory is the so-called *Memory organization packet* (*MOP*). *MOP*s organize situations (cases) with similar activities. A *MOP* usually contains information about normative sequence of events, settings, cast of characters, props that one can expect to see on the situation, and so on. *MOP*s remind people of situations which are in the same problem domain and similar in detailed activities with the new situation (see [42]). By far, most of the existing *CBR* systems use *MOP*s (or its variation) as the model for their memory (case) structures. For more details about *MOP*, we refer to Schank[66].

Before finishing this section, we would also like to point out that, even though we have said a lot about the soundness of *CBR* as a problem-solving methodology, we do not mean that it universally suits for every problem domain. In fact, the *CBR* method is based on two

tenets about the nature of the world (see [47]). The first tenet is that the world is regular: similar problems have similar solutions. Consequently, solutions for similar prior problems are a useful starting point for solving a new problem. The second tenet is that the type of problems an agent encounters tend to recur. Therefore, future problems are likely to be similar to current ones. When the two tenets hold for a problem domain, case-based reasoning becomes an effective reasoning strategy.

## §2.2 What is a case

Cases are the most fundamental objects in a *CBR* system. But what exactly is a case? Generally speaking, a case represents specific knowledge tied to a particular problem context. It usually records knowledge at an operational level, in other words, it makes explicit how a task was carried out, or how a piece of knowledge was applied, or what particular strategies were used for accomplishing a goal, and so on. Cases often capture knowledge that might be hard to capture in a general model. Cases may come in many different shapes and sizes, or cover different time slices. They may represent a problem-solving episode associating a solution with the problem, or describe a scene associating an outcome with the situation, and so on. A common attribute to all cases is that they record an experienced situation that might be different from what is expected. In other words, cases represent existing variations from the norm.

It has to be pointed out that not every situation is worthy of recording as a case. A stored case must teach a *useful lesson* in one way or another. Here, by useful lesson, we mean the information that has the potential to help a reasoner to achieve a goal or a set of goals more easily in the future, or that warn about the possibility of a failure, or point out an unforeseen problem. Based on these observations, Kolodner [41] proposed the following formal definition of a case:

A case is a contextualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner.

What information should be included in the cases for a *CBR* reasoner depends on many factors such as the nature of the problem domain, the goal(s) of the reasoner, the implementation environment, and so on. Kolodner suggested in [41] that, in general, a case should contain the following three major components, although for a particular case they may not all be filled in

- a situation or problem description;
- a solution;
- an outcome;

The *situation/problem description* records the state of the world at the time the case was happening. It might represent a problem that needs to be solved or a situation that need to be interpreted, classified or understood. A case-based reasoner determines whether an old case is applicable to a new situation by examining the similarity between descriptions of the old situation and the new one. If an old situation is sufficiently similar to the new one, that case is selected (retrieved). Thus, a situation/problem description must contain sufficient details to be able to judge the similarity of the case to a new situation. Among many other things, a situation/problem description usually consists of features of the problem situation and the relationships between its parts, goals to be achieved in solving the problem, and constraints on these goals, *etc.*.

The *solution* component of a case is actually the lesson the case can teach when being retrieved. With a solution in place, a reasoner that retrieves the case can use its solution to suggest a solution for the new problem. Solutions can be of various types. A solution can be a derived plan, a designed artifact, an explanation or a classification. In order to aid the reasoner in adaptation and/or critiquing, some other components may also be included in the solution part, for example, the reasoning steps used to solve the problem, justifications

for decisions that were made in solving the problem, alternative solutions, and so on.

The *outcome* of a case specifies what happened as a result of carrying out the solution or how the solution is being performed. Outcomes may include both feedback from the world and interpretation of that feedback. With such information, a reasoner can anticipate potential problems and predict the outcome of a proposed solution. The functionality of a reasoner can be further strengthened by including even more information in the outcome component, for example, whether the outcome is a success or a failure, whether the outcome fulfilled or violated the expectation, an explanation of the violation(and/or failure) of the expectation, a repair strategy, and so on.

Our next concern is how to represent a case, conceptually and physically. It is imaginable that there is no (and actually there should be no) single all-purpose representation schema that would universally suit every problem domain. The most important criteria in deciding a case representation is that the form of representation must be appropriate for the reasoning task of the reasoner. A well-structured case representation should provide the reasoner with effectiveness and time efficiency. For example, some reasoners need to interact with people in one way or another during the reasoning process. Therefore, portions of cases that the machine will use to reason have to be represented in a way that enable that reasoning, while portions of cases that people will use to reason should be represented in a form that a person can easily interact with. As for the physical representation of cases, fortunately, artificial intelligence have provided us with a wide range of representation formalisms (e.g., frames, predicate notations, semantic network, form-like structures, *etc.*), which can be used to physically store the contents of a case in a form that computers can operate on. In fact, case-based reasoners have been built using the whole variety of these formalisms, combinations of them and many other specially designed mechanisms.

Another major issue related to case representation is *case indexing*. In a *CBR* system,

many cases may be stored in its case base for retrieval and reuse, and the effectiveness of the case retrieval determines the performance of the system. Therefore, how cases are organized in the case base is extremely crucial. In a case base, a particular case is identified by its index. The index is an important component of the case representation. It distinguishes one case from other cases. It specifies where a case fits in the entire case base and under what circumstances the case may have a lesson to teach. In other words, it tells when the case is likely to be useful. Indexes are mainly used at retrieval time to judge the appropriateness of an old case to a new situation. According to the above discussion, one should not be surprised by the fact that, a specially structured solution/problem description component of a case is often used to serve as its index.

Under this point of view, indexing must reflect the features, the characteristics of a case. It must represent an interpretation of a situation. The most handy candidates for indexes are the surface features of cases. They are easy to extract and understand. But research and experience (and intuition as well) show that complex combinations and compositions of features serve as much better indexes than simple surface features, because they can distinguish cases from each other by the lessons they teach or by the purposes they fulfil. Though such features usually require computational effort to derive, indexing by these features makes cases more appropriately accessible and allows the retriever to retrieve more useful cases.

Good indexes are those features and combinations of features that distinguish a case from other cases because they are predictive of something important in the case to the reasoning goals. Several guidelines for choosing indexes for particular cases have been proposed (see, e.g. [41]):

- Index should be predictive;
- Predictions that can be made should be useful ones, that is, they should address the purposes the case will be used for;

- Indexes should be abstract enough to make a case useful in a variety of future situations;
- Indexes should be concrete enough to be easily recognizable in future situations.

Application of these guidelines results in indexes that differentiate cases from each other in useful ways, making it possible for the retriever to retrieve cases appropriate to a new situation. Although these guidelines for choosing indexes are each straightforward and understandable, their combination and implementation can be quite complex. Indexes can be chosen either by human or by automated computer programs. In practice, people tend to be better at choosing indexes than any existing index-choosing program, because people have simply better understanding of the situations and are more flexible at the varying situations. In some occasions, however, it may be advantageous for the computer to do the indexing job, especially when problem solving and understanding are already automated.

## §2.3 How does a case-based reasoner reason

Now let us turn to see how a case-based reasoner works as well as some of the key processes in building *CBR* systems, such as case retrieval, case adaptation, case evaluation, and so on. As observed in [5], at the highest level of generality, a conceptual *CBR cycle* may be described by the following four processes (see Figure 1.1, also from [5])

i. *Retrieve* the most similar case(s);
ii. *Reuse* information and knowledge in the retrieved case(s) to solve the new problem;
iii. *Revise* the proposed solution;
iv. *Retain* the parts of this experience likely to be useful for future problem-solving.

In other words, a new problem is approached by remembering (retrieving) one or more old cases from its case-base, moving forward from there, reusing the retrieved case(s) (one way or another) to propose a solution for the new problem, revising the proposed solution according to the available knowledge about the problem domain and the current case, and if necessary, retaining the new experience by incorporating it into the existing case-base. *Figure 2.1* illustrates this whole procedure.

How these processes can be implemented depends mainly on the nature of the particular problem domain that the reasoner deals with. As for *AI* in general, there is no universal *CBR* system building method suitable for every problem domain. The challenge in *CBR* lies in coming up with methods that are suitable for problem-solving and learning in particular subject domains and for particular application environments. In the rest of this

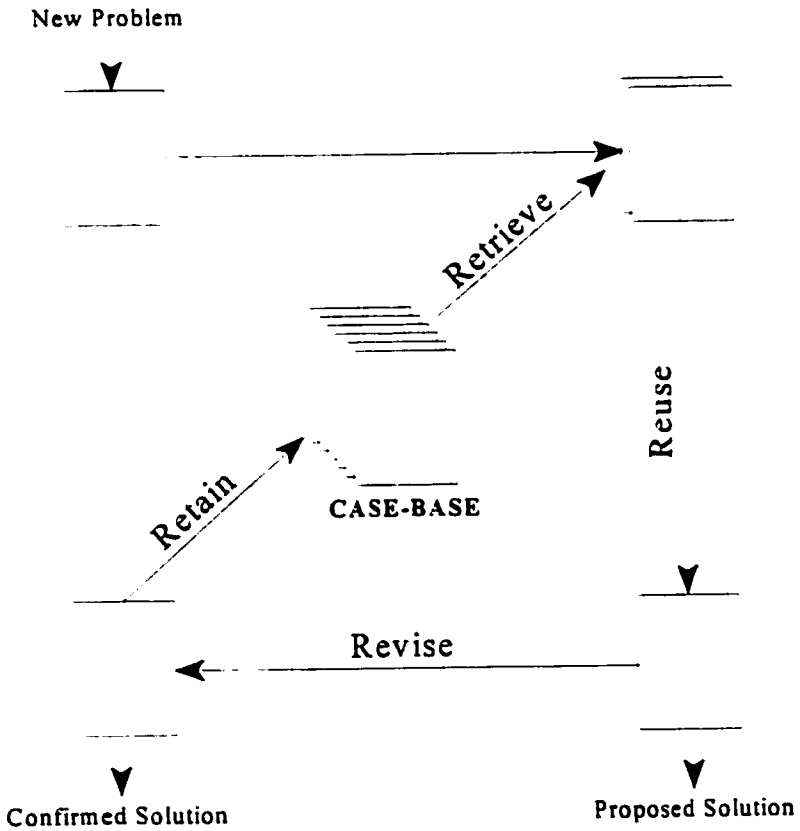


***Figure 2.1*** *The CBR cycle.*

section, we discuss some key technical issues in building *CBR* systems.

**Case Retrieval.** From the above discussion, we have seen that the core component of the case-based reasoning is the case retrieval. Case retrieval is a rather complex process. As described by Kolodner in [41], the (idealized) process goes like this:

> The reasoner asks the retriever for cases by describing its new situation and its current reasoning goal. The new situation is analyzed. Its description is elaborated by situation assessment procedures, which determine what the index for the new situation would be if it were stored in the case library. Retrieval algorithms use the case and computed indexes to search the case library. As they search, they call on matching procedures, either to assess the degree of match between the new situation and cases that are encountered, or to assess the degree of match along an individual dimension. Retrieval algorithms return a list of partially-matching cases, each of which has at lease some potential to be useful in case-based reasoning. Ranking procedures analyze the set of cases to determine which of them has the most potential to be useful. That set is returned to the reasoner.

In order to achieve a satisfactory result, many factors, such as accuracy, efficiency and flexibility, have to be taken into consideration during the design and implementation of the process. In reality, not all those steps described above are implemented for every *CBR* reasoner. For certain problem domains, some of the steps may be simply not applicable or not necessary. For others, some steps may be surrendered in balancing the completeness and efficiency.

One of the most important components of the retrieval process is the *retrieval algorithm*. It is the procedure that knows how to search the case library. Retrieval algorithms are designed in accordance with the indexing structure of the case library. Different organizations of cases give rise to different algorithms for searching a case base. For example, some algorithms search the entire case base while others search only relevant parts of the case base. The same case base may also be searched in different ways for

different purposes.

Before being passed to the retrieval algorithm, the raw information about the new situation is often processed first by another procedure, called *situation assessment*. Situation assessment is the procedure that analyzes a raw situation and elaborate it using the same vocabulary as cases already in the case base. The assessment result tells us in which part of the case library the new case would most likely reside if it is going to be inserted into the library; therefore, the search should be focused on that part of the case library. This process is especially useful when the case base is relatively large, so that the search efficiency can be significantly improved. On the other hand, this assessment result is also valuable for actually storing the new case into the case base when it is decided that the new case is worth being retained.

Another crucial component of the case-retrieval process is comparing individual cases stored in the case library against the new situation and evaluating the degree of match. This task is usually performed by *matching and ranking procedures*. These procedures are very carefully designed so that, with the knowledge of the problem domain and the current reasoning goals, they are able to determine which dimensions inside a case structure are important and which can be safely ignored in determining the degree of match between two cases.

**Case Adaptation.** In case-based problem-solving, old solutions are used as inspiration for solving new problems. Since new situations do not always match old ones exactly, old solutions often need to be modified to fit new situations. This stage of *CBR* is called *case adaptation*. Adaptation plays a fundamental role in the flexibility of problem-solving *CBR* systems, and constitutes the major part of the *case reuse* phase of the case-based reasoning paradigm. There are two main steps involved in adaptation: 1) figuring out what needs to be adapted and, 2) decide what kinds of actions need to be taken.

Difficulties arise from both steps. It is obvious that the need for solution adaptation originates from the difference between the old and new situations. So a sound comparison mechanism is a necessity for identifying those differences. Another important way of spotting what needs adapting is to notice the inconsistences between the old solution and the goal of the new situation. Based on this information, a reasoner could decide which parts of the solution need to be modified, and which parts can be retained as it is.

After knowing what needs to be fixed, we are faced with the question of how to fix it to accommodate the new situation. There could be many possible ways to adapt a solution, but which is the most appropriate one, and how can we control the adaptation process? Answering these questions may require considerable domain knowledge, which in turn raises the question of how to acquire that knowledge. There are some strongly domain-dependent models to deal with this difficulty. But the general adaptation problem still remains extremely challenging for *CBR* system builders. By far, very few existing *CBR* systems can actually do automatic case adaptation without human interaction.

**Case Evaluation.** The case adaptation phase of the *CBR* process suggests a solution for the new problem. This proposed solution is typically tried out in the real world. Feedback about the real things that happened during or as a result of executing the solution is obtained and analyzed by the reasoner. If results are as expected, further analysis is not necessary, but if they are not as expected, explanation of the anomalous result is necessary. This explanation requires evaluating the case solution and figuring out what caused the anomaly and what could have been done to prevent it. It can sometimes be done by case-based reasoning, that is, by reapplying a previous explanation.

This phase of reasoning, called *case evaluation*, is important for a *CBR* reasoner because it gives the reasoner a way to evaluate its decisions in the real world, allowing it to collect feedback that enables it to learn. Feedback allows the reasoners to notice the consequences of its reasoning: this in turn facilitates analysis of its reasoning and

explanation of things that didn't go exactly as planned. This analysis in turn allows a reasoner to anticipate and avoid mistakes it has been able to explain sufficiently and to notice previously unforeseen opportunities that it might have a chance to use later.

Evaluation is the process of judging the value of a proposed solution. Sometimes evaluation is done in the context of previous cases, sometimes it is based on feedback from the real world, and sometimes it is based on simulation. Evaluation includes explaining differences (e.g. between what actually happened and what was expected), justifying differences (e.g. between a proposed solution and one used in the past), projecting outcomes, and comparing and ranking alternative possibilities. Through the evaluation, a reasoner can learn the general situations that will cause failures, and the learnt knowledge can be used in the future case adaptation. Another important role that evaluation plays in *CBR* is that it can point out the need for additional adaptation, or repair, of the proposed solution.

**Case Retainment.** *Case retainment* is the last phase of the *CBR* cycle, which constitutes the learning ability of a *CBR* system. The learning from success or failure of the proposed solution is triggered by the outcome of the evaluation and possible repair. It involves selecting which information from the case to retain, in what form to retain it, how to index the case for later retrieval for similar problems, and how to integrate the new case in the memory structure.

The way new experiences are retained may depend on how the problem is solved. For example, if the new problem was solved by using the solution of an old case, then it is probably more adequate to generalize the old case to subsume the present case than to build a completely new case; if the new problem was solved by a new method (possibly containing some valuable user input), then it might be a better idea to retain the entire case. Some failures may also be worthy of retaining, either as separate failure cases or as part of a complete case. When a failure is encountered in the future, the system can then get a

reminder from a previous similar failure, and use the failure case to improve its understanding of the present failure, and maybe correct it.

## §2.4 How does *CBR* fit our problem

To conclude this chapter, we go back to our own problem domain to argue why we favor *CBR* approach for our system over other approaches, and how *CBR* fit our problem domain. Recall that our goal is to build an expert system that is capable of helping university students in solving their integration problems.

Our first claim is: the integration context satisfies the two tenets for *CBR* methodology mentioned in *Section §2.1*. Recall that the two tenets state: 1) *similar problems have similar solutions*, 2) *the type of problems an agent encounters tend to recur.*

The first tenet may not be true for the integration context if the similarity is defined in the trivial way (*i.e.*, similar in appearance). For example, functions $sin^2(x)$ and $sin^3(x)$ are similar in appearance, but they have to be integrated using rather different tricks. On the other hand, functions $ln(x)$ and $sin^{-1}(x)$ have nothing in common on the surface, but they can be integrated using the same trick. Nevertheless, we can (and we are free to) redefine the similarity in a more meaningful and helpful way to make this tenet true. The details will be discussed in the next chapter.

To see that our problem domain satisfies the second tenet, we just need to notice the fact that the majority of textbooks cover similar contents and techniques in their integration context, although the content might be arranged in different orders. Examples and exercises on the textbooks are usually chosen carefully to reflect the concepts and techniques covered

by the books, so that students may have opportunities to practice what they were taught. When professors assign out-of-textbook exercises, they tend not to stray too far away from the textbook contents as well. Therefore, the pool of problems that first year students could possibly encounter in their *Calculus* course is relatively small. To put this in another way, if there is such an imaginary super professor who teaches *Calculus* in every university across the North America (even the whole world), the problems he might be asked in each school are very likely to fall into one same pool.

Our second claim is: there is no general integration method, which is suitable for human being to perform, but many special techniques (commonly known as *tricks*) are available for solving particular types of integrals. In fact, as we mentioned before, we do have some general algorithms to solve integrals (such as those implemented in *Maple*), but those methods are not suitable for humans, therefore, they do not help students in solving their problem. On the other hand, it is not realistic too to cover all those specific techniques in a textbook or in a classroom. So we believe that a much more practical way to present those specific knowledge to students is to store them in the form of cases, and making them available to students when needed. Here we would also like to mention it again that our system is not intended to be an integration "wizard" that can evaluate any integral. Instead, the system just need to know how to solve certain types of commonly encountered, technically typical integrals.

The third reason why we choose the *CBR* approach is that: those specific integration techniques are quite easy to acquire. First of all, there are some specialty monographs which collect plenty of those techniques (these books are not widely available to students for some reason). Secondly, in our university community, there are many mathematics professors who are proficient and experienced in integration. Last, I personally have plenty of experience in teaching integration.

Our last claim is that: professors actually use *CBR* method all the time when they try

to come up with a hint for students' integration problems. For the majority mathematicians, their knowledge in integration is in fact in the form of cases (*i.e.*, specific experiences). Only those very few mathematicians who are specialized in the integration algorithms may have the knowledge of those very general integration methods. As we have observed and analyzed in *Chapter One*, when a professor faces an integration problem, he/she seldom use any general rule to deduce a solution. Instead, his/her solution comes from his/her past experiences in solving integration problems.

All in all, we believe *CBR* is the right problem-solving and learning model for our proposed tutorial system.

# Chapter Three

# System Design and Implementation

In the previous chapters, we discussed the problem we are going to tackle in this project and the problem-solving approach we are going to take. Now, we present a more detailed picture of our system, from both internal and external point of view. In this chapter, we discuss how the chosen methodology is being materialized, how the system is physically implemented and how the system interacts with users. The first section is a brief overview of the whole system. In second section, we discuss the case structure in our system, which is the core object for a *CBR* system. As you will see later that our system is physically divided into two major parts, the client-side component and the server-side component. The next two sections will focus on these two parts respectively. The last section will discuss some issues related to our searching and matching algorithm.

## §3.1 An overview of the system

Recall that, this system is intended to be an intelligent system which is capable of helping university students in solving integration problems. Due to the problem nature, as discussed in the previous chapters, we decide to adopt the case-based reasoning approach as the problem-solving method for this system. As a *CBR* system, the reasoning resource

(the knowledge) of this system is mainly in the form of cases. Each case in our case base represents a real integration problem-solving experience (including the problem itself, the hint about how the problem can be solved and the answer). Therefore, this portion of knowledge is very problem-specific. We will discuss our case structure in the next section.

It is not difficult to see that our case base provides students similar type of knowledge as textbook examples do. But, there are some essential distinctions between the two. First of all, the case base has nearly no volume limitation. It can hold as many cases as necessary. This fact greatly increases the chance for students to find examples that may help them solve their own problems. Secondly, unlike textbook examples (and cases in most of other *CBR* systems), the problem solution is excluded from our cases. This decision is based on and justified by following two observations: 1) it is a much more effective way to learn for students to carry out the solution first-handed than just copy a solution; 2) the majority of students are capable of carrying out the solution on their own with the help of hints.

The third distinction between our case base and textbook examples is that, with a textbook, students have to search similar examples using their own judgement on the similarity which is often not well-defined. On the other hand, our system automates this example searching process with the experts' insight. For this purpose, the system is also equipped with another important knowledge source, besides the case base. This part of knowledge is implicitly implemented into the searching and matching module, in order to help the system to identify similar cases. This part of knowledge is in the form of rules. It consists of a number of general experiences in solving integration problems. As we mentioned in *Chapter One*, it is *not* appropriate to simply define the similarity along with the appearance in our problem domain, because it does not guarantee the truthfulness of one of the *CBR*'s tenets: similar problems have similar solutions. So, with the help of these general experiences, our system gains the ability to avoid retrieving those cases which are similar to the new problem in the surface features but the methods used to solve them do not apply to the new problem. In other words, these general experiences define a more

appropriate and sophisticated similarity. We will conduct a more extensive discussion on this matter in a later section.

Now, let us take a look at the design aspect of the system. Recall that one of the main objective of this system is to complement the unavailability of instructors at the time help is needed. So, our system must be accessible to any student anywhere anytime. The current internet technology provide us an excellent platform to reach this goal. For the maximum accessibility, the system is designed and implemented as an internet application, using various internet techniques. The system is implemented using the native internet programming language, *Java*, as a *Java applet* so that it can be accessed through any Java-enabled web browser. Nowadays, the internet access is no longer a privilege or a luxury, especially for university students. With the free on-campus internet access and unexpansive commercial accesses, the Internet has become one of the most important information source for university students.

As we know, a Java applet is a Java program that is usually embedded into a web page. It is downloaded to and run on the user's machine when the embracing page is accessed. But, due to two major concerns, efficiency and security, we do *not* want our case base to be sent to the user end. One reason is that the volume of our case base is relatively large. From the time efficiency point of view, it is not a good idea to send it to the user end through the ever-crowded internet. On the other hand, to ensure the integrity of the system, we also do not want to give anyone any chance to tamper with our case base. Therefore, the case base has to be physically separated from the rest of the system and stay in the server permanently. This gives rise to a question: how do the parts of our system, which work on different sites on the Internet, communicate with each other? Our solution is the *Common Gateway Interface* (*CGI*) program interface model.

*CGI* is a standard for information servers (such as a Web server) to interface with external applications. More specifically, with the *CGI* mechanism, a server can invoke, pass

information to and receive response from external applications. A very popular use of *CGI* is *HTML form*. When a "*Submit*" button in a form in a web page is pressed, the web server in which the page resides typically invoke an external program to process the information on the form (filled by the site visitor). In most situations, the external program returns something to the server (for example, a dynamically generated web page). Upon the reception of the response from the program, the server relays it to the site visitor. Those external programs are commonly known as *CGI scripts*, and they usually reside in a special place in the server, called *CGI-bin*.

Now, back to our system. For the efficiency and security concern, our system is physically divided into two parts (see *Figure 3.1*). One is called *server-side component*, the other one is called *client-side component*. The server-side component, which resides in our web server permanently, consists of the case base and its *public interface*. The public

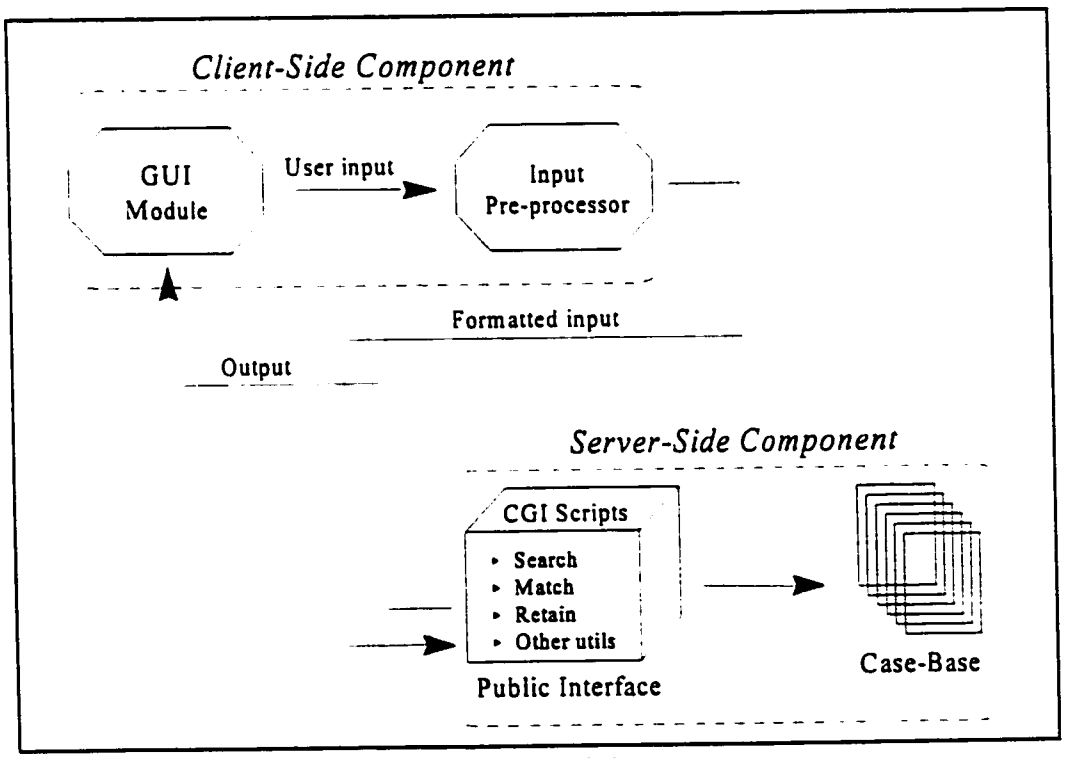


***Figure 3.1***

interface of the case base is actually a group of *CGI* scripts, which are used to access and maintain the case base. In order to maintain the integrity of the case base, the access to our case base is restricted only through its public interface.

The client-side component is a Java applet. Conceptually and logically, it can be divided into two parts as well. One is the *GUI* (Graphical User Interface) *module* which constitutes the user's view of our system. In other word, the system communicates with users through this *GUI* module. It takes user inputs and displays system responses on the screen. It also monitors user actions (such as confirming important actions, warning dangerous actions and reject meaningless actions, *etc.*). The other part of the client-side component is the so-called *input pre-processor* which performs certain preliminary input data processing duties behind the scene. This module provides a data structure for storing and manipulating user inputs. It also validates and transforms raw data from users into the internal format which the other parts of the system can understand.

At run time, the client-side component runs on the user's machine, while the server-side component stays on the home server. The two components communicate through the *CGI* mechanism implemented on the server. When a request is made to the system through its *GUI* module, the server creates a channel connecting the *GUI* module and the public interface of the case base. Then, one or more relevant *CGI* scripts will be triggered to process the request. The input to and output from the scripts are both transmitted through this channel. When the *GUI* module receives the response from the script, it displays the information on the screen in a proper way. After the "talk" (data exchange) is over, the channel is closed by the server and the two components are separated until the next request is made by the user.

## §3.2 The case structure

As we have discussed in *Chapter Two, CBR* systems are extremely case-centric. Every components of a *CBR* system is built around its case base. Our system is no exception. In this section, we will discuss how a case is structured conceptually and physically in our system.

Our system is intended to be an "expert" in solving integration problems. Recall that, in our problem domain, each problem is represented by an individual integral, therefore, each case in our system is structured around an integral. Conceptually, each case is the combination of its external view, its internal view and its solution. Physically, a case consists of the following elements:

- visual image of the integral,
- a textual description of the integrand ,
- useful hints about how to solve the integral,
- an answer to the integral.

The last two items in the above list, the hints and answer to an integral, constitute the solution part of our case (*cf.* §2.2). Of course, hints are not quite a solution to an integral problem, but it tells the way how a solution can be deduced. We intentionally exclude the full solution from our cases in order to encourage students to carry out solutions on their own effort, with the help of the hints. The inclusion of answer is intended to help students in judging if they have carried out the solution correctly. The answer also helps students in another way that it often provides them some extra hints on how to apply the given hints. For example, with an answer in hand, students may know what should be the goal of their efforts, and make their attempts more purposeful.

The image of the integral represents the external view of the case. In other words, it is the way how users of the system see the case. The image is in the *jpg* format, generated by $TEX$ and some image tools. All the images are stored on the server in which the system

resides. They are loaded to the client-side and displayed in the designated area on the *GUI* only when needed (say, selected by the search engine). A typical image of an integral shows an integral just like it is being showed on any textbook, here is a sample:

$$\int \frac{x}{\sqrt{(x^2-a^2)^3}}dx$$

There are other ways to represent integrals. For example, in the $TEX$ syntax, the above integral can be expressed by the following string:

```
\int \frac {x}{\sqrt {(x^2 - a^2)^3 }} dx
```

It could have saved us a significant amount of storage space had we used this type of textual representation as the external view of our case. But we choose the graphical representation over the textual one for the following reasons. 1) The graphical representation is much easier to recognize by users than the textual representation. Our system is an interactive system which expects users who are most likely not very experienced in mathematics. Typically, during a run, the system responds a user request with a list of integrals, and then the user will decide if any case on the list matches his/her problem, if he/she would like to see the hint for any integral, or if he/she would like to request a further search. The user's decision is mainly based on the comparison of the user's problem against the cases on the list. Therefore, the graphical representation makes this crucial step much easier for users. 2) On the current market, there is no internet browser that is capable of translating any type of textual representation (whether in $TEX$ syntax, or in *Maple* syntax, or in any other format) of a mathematical formula into graphical representation. *HTML* version 3.0 has included a *<MATH>* tag in its standard which is supposed to allow browser venders to implement this kind of translation. But by far, it is still nearly a hollow tag which is ignored by all browsers. 3) With any currently existing mathematical formula textual representation syntax, the representation of an integral is not uniquely determined. In other words, one

integral can be expressed in more than one ways using the same syntax. This may cause even more confusion to students.

The most import element of all in our case is perhaps the second element, the textual description of the integral. This description serves as the internal view of the case, *i.e.*, how the system itself sees a case, or how a case is known by the system. The essential role of this description is to act as the index for the case used for organizing the case base.

As we have mentioned previously, there are several existing powerful syntactical languages that are capable of expressing integrals using pure text (for example, $T_EX$ syntax, *Maple* syntax, *etc.*). These languages are logically very precise and syntactically flexible. However, they are not suitable for our system for the following three reasons.

1) The main reason is that these syntaxes are of too much "graphics-oriented". They are designed to describe the graphical appearances of mathematical expressions textually, so that all the mathematical formulas can be stored in text files and be dynamically translated into graphics. As we all know that most of mathematical functions are combination or composition of the so-called basic functions. The components of a function can be placed in different orders or different places. In other words, one function may be graphically expressed in more than one ways. For example, the integrands of the following two integrals are mathematically identical, even though they appear differently:

$$1)\ \int \frac{\sqrt{a^2+x^2}}{x^2}dx \qquad\qquad 2)\ \int x^{-2}\sqrt{a^2+x^2}\,dx$$

Therefore, to express a mathematical expression in $T_EX$ syntax (or any other existing syntaxes), how one likes it to appear graphically is very crucial. In other words the $T_EX$ description of a mathematical expression depends on the desired graphical position of its components. For example, in $T_EX$ syntax, the above two graphical expressions of the same

integral are expressed rather differently:

```
1)  \int \frac{\sqrt {a^2 + x^2}}{x^2} dx
2)  \int x^{-2} \sqrt{a^2 + x^2} dx
```

But, to our system, what is important about an integral is how it can be evaluated, not how it should be displayed graphically. Therefore, the graphical order or position of components in a function is irrelevant to our problem domain, and so, it should not take part in the case index for our case base.

2) The second reason is that, syntactically, the $T_EX$-like syntaxes are too flexible. By "flexible", we mean that each graphical expression of a mathematical formula can be described in many different ways. For example, in the $T_EX$ syntax, the first integral above can be described by at least three different strings shown below:

```
a)  \int \frac{\sqrt {a^2 + x^2}}{x^2} dx
b)  \int \frac { \sqrt{a^{2} + x^{2}} } {x^2} dx
c)  \int {\frac{\sqrt {a^2 + x^{2}} } {x^{2}} dx
```

Syntactical flexibility may be good for the people who use the languages to compose mathematical formulas because it makes their work much easier. But, from a system builder's point of view, it may not be something to cheer for, because it makes the parsing of expressions very difficult. It may take a human (who knows the $T_EX$ syntax) very little effort to recognize that these three string describe the same integral, but it will take a system builder a tremendous amount of effort to teach the system to do the same.

3) The $T_EX$-like syntaxes do not help us in defining the desired similarity. As we mentioned in *Section §2.4*, the similarity between integrals in our system can not be defined simply along with their appearance, because the similarity in appearance does not guarantee

the similarity in the solution, and *vice versa*. For example, in our system, we would like the following pair of integrals to be regarded as "similar",

$$1) \int \cos x \sin^5 x \, dx, \qquad 2) \int \sin x \cos^n x \, dx;$$

because they can be solved using very similar methods. If one was taught how to solve one of them, he/she would be able to solve the other with no difficulty. But one can hardly say that the following $T_EX$ expressions of those integrals are textually similar in any sense:

```
1) \cos x\sin^5x
2) \sin x\cos^n x
```

Therefore, we need to find a different way of expressing (or indexing) integrals, which captures the components of a mathematical function but ignores their graphical positions, which can be easily understood by a computer program (*i.e.*, can be easily implemented), which helps us define the desired similarity, and all in all, which is suitable for the problem-solving methodology adopted by our system.

Instead of regarding an integral as a whole, our strategy is to break it (the integrand, to be more precise) down to the *factor* level, that is to say, we consider any integral as a product of one or more of its factors (a factor of a function is such a component of the function that the entire function can be expressed as the product of this component and another function). As for how to partition an integrand into factors, we do not impose any specific rule on this procedure. Rather, we suggest to follow some common-sense rules. Here are two of them:

- a function should be sliced along with the basic functions it contains
- a factor should contain as few different types of basic functions as possible

According to these rules, function $\sin x \cos^2 x$ should be partitioned into $\sin x$ and $\cos^2 x$, rather than $\sin x \cos x$ and $\cos x$. In most situation, a factor should contain no more than one type of basic function. But there are exceptions, say, the factor $\sin(\ln x)$, can not be split any more, even though it consists of two different types of basic functions.

As we mentioned many times, we do not care the relative graphical positions of the factors contained in a function. This fact allows us to express (or indexing) individual factor independently of others. Then, naturally, the expression for the whole integrand itself should evidently be the collection of all the expressions for its factors, without particular order.

To facilitate the definition of a desired similarity, we categorize factors into groups according to the types of underlying basic functions. Our system, as an experimental system, currently supports only three categories of factors: *trigonometric factors, quadratic factors* and *polynomial factors*. Trigonometric factors include factors consisting of trigonometric functions such as *sin*, *cos*, *tan* and *cot*. For example, following functions are all considered to be in the trigonometric factor category:

$$\sin x,\ \cos x,\ \sin^2 x,\ \frac{1}{\cos x},\ \sqrt{\tan x}.$$

Note that, for the simplicity, *sec* and *csc* functions are not included in our system (but it will be a simple and straightforward extension to include them). Quadratic factors include factors consisting of expression of the form:

$$ax^2 + bx + c \quad (where\ a > 0).$$

Polynomial factors include factors consisting of expressions of the form:

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \quad (where\ a_n > 0).$$

To index factors, we assign each category a formatted string variable. The format differs from category to category. Each byte in a string variable serves as an indicator of certain mathematical properties relevant to the corresponding category. Some of the common properties captured by all three categories are: whether the factor is in the denominator, whether the factor is under root, what is the power the underlying basic function is raised to, *etc.*. There are also some category-specific properties. For example, type of the underlying basic function is captured for trigonometric factors (whether it is a *sin* , *cos*, or *tan*, *etc.*), the +/- sign for the x-square term is captured for quadratic factors. These properties are monitored because they make significant differences in the method used to evaluate integrals. The possible values that could be assigned to a byte indicate the correlations between the described factor and the underlying basic function in the designated mathematical property.

The index of each factor is determined by the category it belongs to, the corresponding formatted string variable and the relation between the factor and the underlying basic function. The value of the index is obtained by filling the individual byte in the string variable with appropriate values. Here are some examples:

| | |
|---|---|
| $\sqrt{sin^5 x}$ | *trig sin* 000000 5 *n y* 2 1 1 |
| $1/(1+sin^2 x)^2$ | *poly sin* 2 *y n* 1 2 1 |
| $\sqrt{(a^2 - x^2)^3}$ | *quad a -x* 3 *n y* 2 1 |

we will not discuss the specific meaning of each byte in the index here. For interested readers, we refer to *Appendix A*.

The index of a function is simply the concatenation of indexes of all its factors without specific order, separated by colon (:). For example, the index of function $x^2 sin^2 x \, cos x$ could be either one of the following:

| |
|---|
| 1) poly _x_ 2 n n 1 1 1:trig sin 000000 3 n n 1 1 1:trig cos 000000 1 n n 1 1 1 |
| 2) poly _x_ 2 n n 1 1 1:trig cos 000000 1 n n 1 1 1:trig sin 000000 3 n n 1 1 1 |

Apparently, indexes of factors have to be concatenated in some order physically. But what we mean by "without specific order" is, no matter how they are concatenated, the system treats them as un-ordered internally.

To conclude this section, we would like to present a complete case of our system, to illustrate what we have discussed in this section.

| Image | $\int x^2 sin^2 x \, dx$ . |
|---|---|
| Index | 25:2:poly _x_ 2 n n 1 1 1:trig sin 000000 2 n n 1 1 1:*570 |
| Hint | First, try Integration by Parts:<br>u = x^2, dv = sin^2(x)<br>To integrate dv, you need to apply the formula:<br>sin^2(x) = (1 - cos(2*x))/2<br>Another similar (but simpler) Integration by Part may be needed after the first step. |
| Answer | (x^3)/6 - (x^2)*sin(2*x)/4 + sin(2*x)/8 - x*cos(2*x)/4 |

Here are two remarks on the above example. 1) In the index, the first number is the internal identifier of the case, while the second is the number of factors the functions is split

to. This information is actually nonessential to the case, it is included in the index simply for the performance consideration. The number appended to the end of the index (prefixed with *) is the physical location of the hint/answer in the hint file. As we will see later, in our system, indexes and hints/answers are stored in separate files. For the speedy access of hint/answer content, we append the location of hints to the end of the index. But logically, it is not part of the index, because our system's search engine ignores it. It is only used when a user decides to see the hint/answer of the integral. 2) The mathematical expressions in the hint/answer are expressed in the *Maple* syntax. In our original design, they were also expressed in graphics. But later, we decided that graphical expressions are neither space nor time efficient, and *Maple* syntax is quite self-explanatory and easy to understand by users.

## §3.3 The client-side component

As we mentioned in *Section §3.1*, the client-side component is implemented with Java as a Java applet. The program is imbedded into an *HTML* file (a web page). When the page is accessed from a remote site, the program will be downloaded to and run on the client's machine.

Conceptually and logically, this component can be divided into two major parts. One is what we called the *input pre-processor* which performs certain preliminary input data processing duties on the back scene. This module consists of four Java classes. From the structural point of view, this module provides several crucial data structures for storing and manipulating user inputs. It also validates and transforms raw data from users into the internal format which the other parts of the system can understand. In other words, this module is where our indexing schema is implemented. From the functional point of view, this input pre-processor provides the system with the following two major functionalities. Firstly, it gives the system the ability to know where a new problem fits in our case base before the search procedure starts, so that the search engine need only search part of the case

gbase and safely ignore the rest of it. This ability virtually partitions our case base into sections and makes the search procedure more efficient. Secondly, it helps system administrators to formulate new cases and store them into the case base appropriately, when they decide the cases are worthy to retain.

The other major part of the client-side component is the *GUI* (*Graphical User Interface*) *module* which constitutes the user's view of our system. In other word, the system communicates with users through this *GUI* module. It takes user inputs and displays system responses on the screen. It monitors user actions, by performing duties such as confirming important actions, warning dangerous actions and rejecting meaningless actions, *etc.*. It is also responsible for invoking appropriate *CGI*-scripts in responding to users' request to access the case base, and to load and display integral images representing the selected integrals. This module consists of twelve Java classes. They implemented all the visual parts of the system such as the main interface window, the input window, information displaying windows, the maintenance utility window, and several message windows.

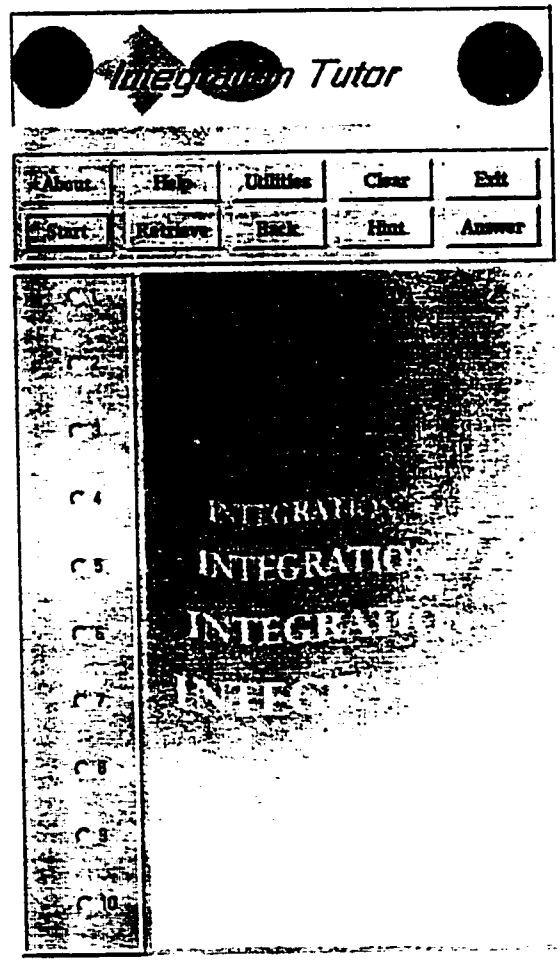


***Figure 3.2***

*Figure 3.2* is a snap shot of the main interface window of our system, which is the center piece of the entire *GUI* module. The upper portion of the window is a button panel holding ten buttons representing various functionality of the system. The lower portion of the window is

a display area. The system displays the retrieved integral images in this area.

In a typical run of our system, a user starts a search session by clicking the "*Start*" button. Then the window showed in *Figure 3.3* pops up. This screen is in fact the "cover page" of a deck of "forms". Each form is designated to collect information about factors of one category(see *§3.2*). Clicking on one of the radio buttons will bring up to top the corresponding form (currently, only top three buttons are active). Each form contains several relatively self-explanatory and straightforward questions. For example, *Figure 3.4* shows the questionnaire for the quadratic factors. The user is expected to specify his/her problem to the system factor by factor, by filling out the corresponding forms. At any point of this process, the user can choose to display and modify currently saved inputs, or to clear all the inputs and start over again. When the user thinks that he/she is done with the problem specification, he/she can ask the system to search its case base to find some examples (cases) similar to his/her problem by clicking the "*Start Search*" button.

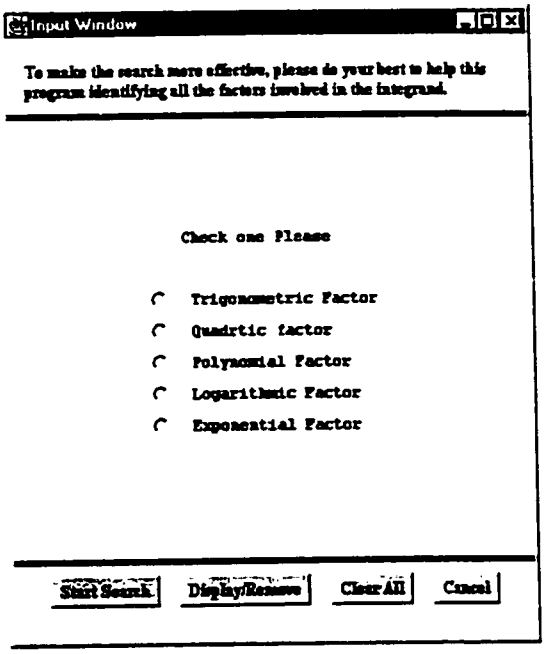


*Figure 3.3*

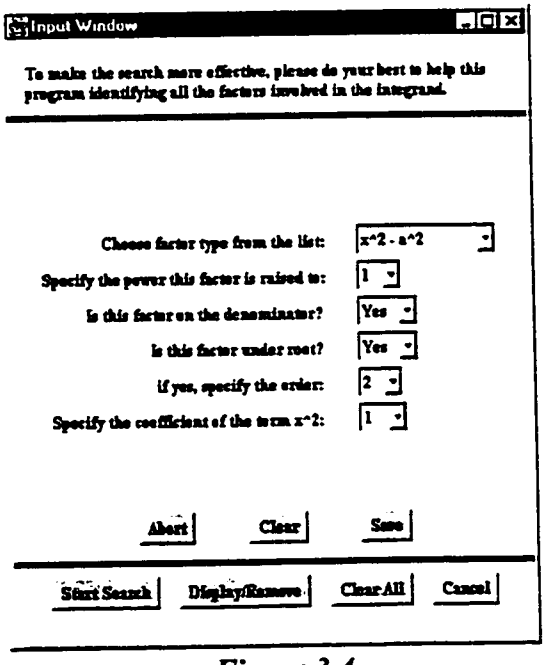


*Figure 3.4*

When the system gets the "*Start Search*" command, it first passes the raw

inputs to the input pre-processor. The pre-processor processes the inputs and returns a formatted index string for the user specified new problem. The *GUI* module then passes the index to the search engine (a *CGI*-script) on the server-side through the Internet. The search engine searches the case base using the given index as the search key and returns the internal identifiers of a list of candidate close-matching cases to the *GUI* module. Finally, the *GUI* module loads and displays the corresponding images on the display area on the main window. *Figure 3.5* illustrates the search result when the function $x^2 sin^2 x$ is given as the new problem.

If the user find his/her problem on the list, or do not find his/her problem on the list but find one which is very close to his/her problem, he/she can view the hint and answer for the integral by selecting the integral first and then clicking the "*Hint*" and/or "*Answer*" button to retrieve the information. The information is displayed on the screen showed in *Figure 3.6*. The content on this screen can be flipped between the hint and the answer by clicking the bottom-left button on the window.

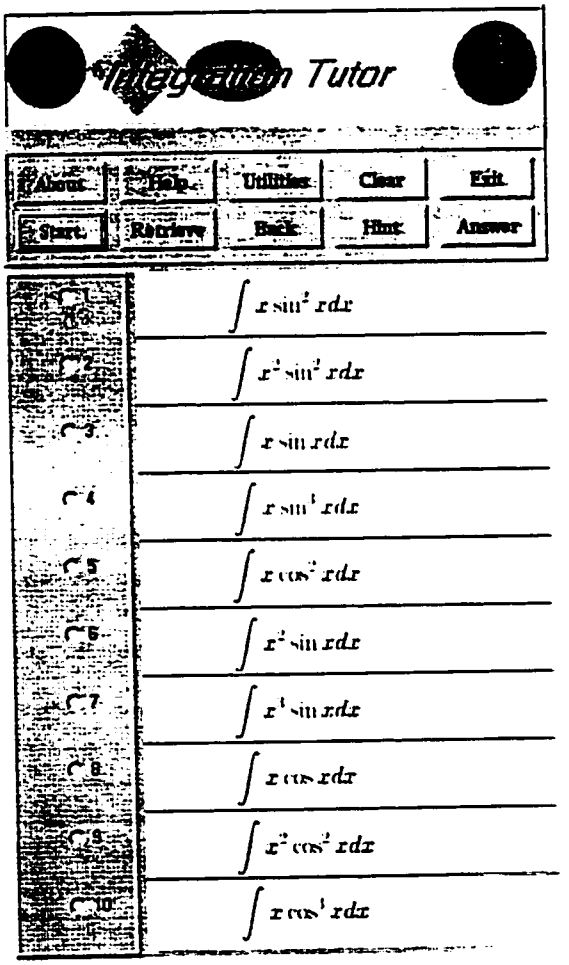


***Figure 3.5***

If the user feels that one of the integral is close to his/her problem but not close enough he/she can request a further search around that integral by selecting it and then clicking the "*Retrieve*" button. This time, the search engine searches the case base

using the index of the selected integral as the search key, and returns another list of close-matching cases. In case the user feels that none of the integrals on the list matches his/her problem, he/she can also choose to start a fresh search by re-specifying his/her problem to the system, maybe in a slightly different way. The "*Back*" button can be used to go back to the previous search result (but the system only remembers the immediately previous result).

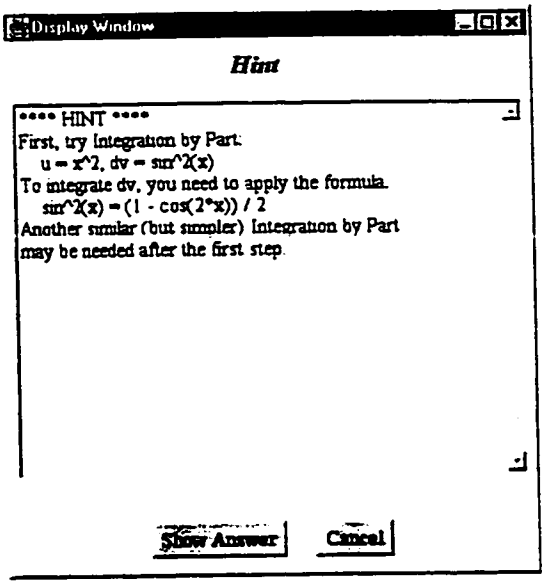


*Figure 3.6*

Now, we turn our attention to the system administrator's view of the system, which is quite different from that of student's. The system administrators' responsibility is to maintain the system, especially the case base. Occasionally, they may find the need to adjust the index of an integral, or to fix the hint/answer content of an integral, or to add an alternative method to the hint, or to add a completely new case to the case base. Therefore, they require some functionality in addition to those available to students. For their convenience, we implemented some system maintenance utilities to automate these tasks. These utilities can be accessed through the *System Maintenance Utilities Window* showed in *Figure 3.7*. For the safety reason, this window is password-protected because, obviously, we can not afford to allow any unauthorized person to access our case base.

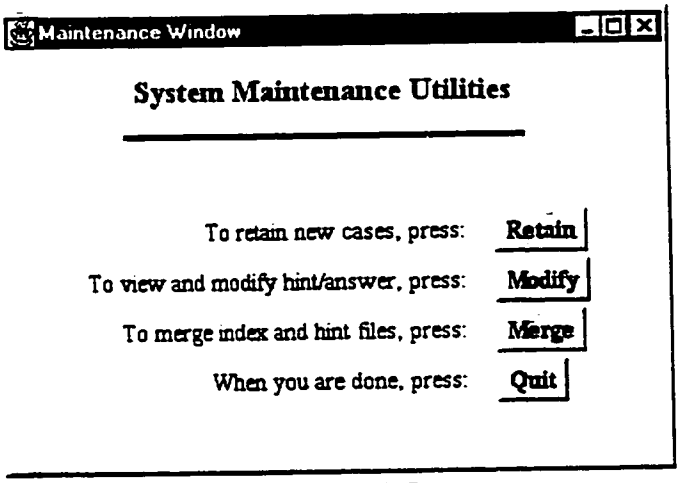


*Figure 3.7*

The "*Retain*" utility provides the system with the learning ability. When a system administrator finds a good example, which is not covered by the current case base, he/she can add it to the case base using this utility.

When the "*Retain*" button is pressed, the input window as shown in *Figure 3.3* pops up to take the specification of the new integral. The only difference is, on this window, the bottom-left button "*Start Search*" is replaced by the "*Save To File*" button. When the "*Save To File*" button is pressed, the system saves the index for the new case in the index file. At the same time, a hint/answer template is also created in the hint file. The actual hint/answer content can be added later using another utility, as shown in *Figure 3.8*. This window is obtained by clicking the "*Modify*" button on the utility window. With this utility, one can view and modify the hint/answer content of any existing case. Therefore, to add the hint/answer content for a new case, one just need to edit the hint/answer template for that case.

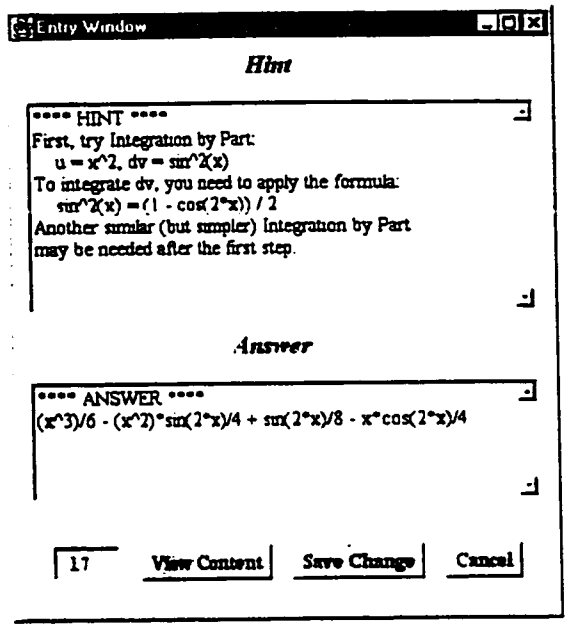


*Figure 3.8*

The "*Merge*" button is used to incorporate the changes to the hint file into the index file. As we mentioned before, in our system, indexes and hints/answers for cases are stored in separate files. For the speedy access of hint/answer content, the physical location of the hint/answer content for a case in the hint file is attached to the end of its index stored in the index file. Therefore, whenever the hint file is modified, the physical location for some cases will change, hence the corresponding information in the index file becomes out of date. To ensure correct content to be retrieved, we "merge" two files every time we edit the hint file.

## §3.4 The server-side component

Unlike the client-side component, the server-side component resides in and runs on the home server permanently. Conceptually, this component can be also divided into two parts. One is the *case base*, the other is the *public interface* of the case base (see *Figure 3.1*). The case base is the knowledge repository of the system, and the public interface is the "*tool box*" that contains all the necessary tools for accessing and maintaining the case base.

First, let us look at the case base. The case base consists of two textual files, *index file* and *hint file*, and all the integral images (representing the external view of cases). Images are stored in the same place in the server where the client-side component resides. But, they are not automatically loaded to the client's machine when the system is invoked. Only those images representing the selected cases by the search engine will be loaded by the system dynamically.

Two textual files are stored in the *CGI*-bin directory in the server, since they are to be accessed by *CGI*-scripts. The hint file contains the hint/answer contents for all the cases in our system. The system guarantees that every case has an entry in the hint file. Even for a newly added case, the system creates a hint/answer template for it as a placeholder, so that the actual hint/answer content can be inserted later. Notice that the size of the hint file could be quite large, because there are already hundreds of cases and an unlimited number of new cases could be add down the road. Therefore, fast and accurate access to the content of any particular case is very crucial to the performance of the system. For this concern, the hint/answer content for each case is formulated into a fixed format, as shown in *Figure 3.9*. This fixed format also makes the maintenance job much easier. To enhance the learning ability of the system, we also allow the hint/answer content for each case to be modified. But, in order to preserve the reliability and the format of the content, the content can be modified only by the authorized person, and only through the provided tools.

```
#---- 17
**** HINT ****
First, try Integration by Part:
   u = x^2, dv = sin^2(x)
To integrate dv, you need to apply the formula:
   sin^2(x) = (1 - cos(2*x)) / 2
Another similar (but simpler) Integration by Part
may be needed after the first step.
~~~~
**** ANSWER ****
(x^3)/6 - (x^2)*sin(2*x)/4 + sin(2*x)/8 - x*cos(2*x)/4
*END
```

*Figure 3.9*

The index file collects the indexes of all the cases. In the index file, indexes are organized into lines, *i.e.*, each line of text contains exactly one index. *Figure 3.10* shows a

```
25:2:trig sin 000000 2 n n 1 1 1:poly _x_ 1 n n 1 1 1:*324
26:1:poly sin 2 y n 1 1 1:*375
27:2:trig cos 000000 1 n n 1 1 1:poly _x_ 1 n n 1 1 1:*446
28:2:trig sin 000000 1 n n 1 1 1:trig cos 000000 2 n n 1 1 1:*532
29:1:trig sum 110000 2 y n 1 1 1:*624
30:2:quad x-a 1 n y 2 1:poly _x_ 1 n n 1 1 1:*706
```

*Figure 3.10*

chunk of indexes from the index file. Notice that, each index starts with two integers and ends up with another one. The first is the internal identifier of the corresponding case. The second is the number of factors in this integral. The integer appended to the end of the index is the physical location of the hint/answer content in the hint file for the case. Logically, these elements are not considered as part of our case index, because they are ignored by the

system's search engine (therefore one set of numbers can be assigned to any case without affecting its logical location inside the case base). They are included in the index purely for the performance reason. For example, when a case is selected as a close match to the new problem by the search engine, its internal identifier and its hint/answer location in the hint file will be transmitted to the client-side component. The *GUI* module will uses the identifier to load the corresponding image, and uses the hint/answer location to retrieve the hint/answer content when the user requests it.

Now, let us turn to the public interface of the case base. The public interface consists of a set of five *CGI*-scripts. They are: *searchcb.pl*, *retain.pl*, *modhint.pl*, *merge.pl* and *fetchint.pl*. As one can tell from the extensions, they are all written in *Perl*, the most popular scripting language for *CGI*-script. These scripts reside in the same *CGI*-bin directory on the server as two textual files mentioned above do. Among these scripts, *searchcb.pl* is the most significant one. It constitutes the search engine of the system. It implements our searching and matching algorithms. The main functionality of this script is to conduct the case retrieval, which is the core of the case-based reasoning paradigm. We will discuss a little more details about our search engine in the next section. The script *Fetch.pl* equips the system with a handy tool to fetch the hint/answer content of a particular case upon the request. The other three scripts are case base maintenance utilities. They help the system to retain new cases, to modify existing hint/answer contents or insert new hint/answer contents, and to coordinate the components of the case base. For short, they provide our intelligent system with the learning ability, which makes the system more and more intelligent down the road.

## §3.5 The search engine

To conclude this chapter, we briefly discuss the search engine of our system and the searching and matching algorithms implemented in the search engine. The searching and

matching algorithms are the core of the search engine which, in turn, is the core of our system. Conceptually, the search engine consists of two main modules: the searching module, which implements the searching algorithm, and the matching module, which implements the matching algorithm. It would be tedious to discuss every detail of the algorithms. Therefore we will only present some of our main strategies in the algorithms.

Let us look at the searching module first. Our searching strategy is quite simple and straightforward. The search key of each search is the index of an integral. This integral could be a new problem or an existing case in the case base. The search engine compares this key index with each entry in the index file by calling the matching module. The matching module takes two indexes and returns a nonnegative integer, called the *degree of match*. The degree of match reflects the system's judgement on how close the two integrals are. The larger the degree of match is, the closer the system believes the two integrals are. During each search session, the search engine keeps track of ten integrals which have the highest degree of match to the key index. Before the search engine terminates, it sends the internal identifiers of these selected integrals to the *GUI* module which requested the search through the *CGI* mechanism. Upon receiving the list, the *GUI* module loads and displays the images of the corresponding integrals to the user.

The matching algorithm is much more complex. First of all, let us notice that there are possibly more than one factor in an integral. When a user specifies an integral, he/she has to enter the factors one by one in some order, and also the system has to put the indexes of all factors together in some order, in order to store them physically. But, on the other hand, it is a common sense that the order of the factors plays no role whatsoever in deciding how the integral should be evaluated. Therefore, to eliminate the influence of the order in which the factors are placed in the index, the comparison must be conducted in the factor index level, instead of the integral index level. For this purpose, the matching module first chops two given integral indexes into two sets of factors. Two sets are then sorted and stored in two arrays. The sorting, which is based on the type of factors, is purely for the technical

purpose. We have by no means any intension to favor one type over the others. In fact, our algorithm does not mind in what order the factors are sorted, as long as the factors in two indexes are sorted in the same way.

The next step is to compare the two integrals. There are two levels of comparison. The lower level comparison is to determine how close any two factors (one from each integral) are. The result of this comparison is a nonnegative integer, called the *degree of match* of two factors, which reflects the system's judgement on how similar two factors are. If the degree of match is nonzero, then the system thinks two factors match to some extent, and the larger the degree of match is the closer the system believe they are. The lower level comparison is conducted by a procedure called *compareFactors*, which implements a substantial amount of general knowledge about how to determine the similarity between two factors (we will discuss this in details later in this section).

The higher level comparison is to determine how similar two integrals are, by finding if any factor in one integral has a matching factor in the other integral. The similarity between two integrals is also measured by a nonnegative integer, called the *degree of match* of two integrals, which is (roughly speaking) the sum of the degrees of match between matching factors of two integrals. Therefore, the higher level comparison actually relies on the results from the lower level one. The algorithm for the higher level comparison is listed in *Figure 3.11*. Here we suppose *IndexA* and *IndexB* are indexes for two integrals to be compared, and suppose *FactorSetA* and *FactorSetB* are the sets of indexes of factors corresponding to *IndexA* and *IndexB*, respectively.

Notice that, in each iteration of the "*for*" structure, the *FactorSetB* might be different from the one in the previous iteration, because a factor in *FactorSetB* will be deleted from the set if it is found to match a factor in *FactorSetA*. This strategy is intended to avoid the situation when one factor in *FactorSetB* is used to match more than one factors in *FactorSetA*.

```
degreeOfMatch = 0;
for each factorA in FactorSetA {
   compare factorA with each factor in FactorSetB, by calling the
      lower level comparison procedure compareFactors, to find the one,
      say factorB, that yields the highest degree of match, denoted by
       partialDegreeOfMatch;
   degreeOfMatch = degreeOfMatch + partialDegreeOfMatch;
   If (partialDegreeOfMatch > 0)
      drop factorB from FactorSetB;
}
return degreeOfMatch;
```

***Figure 3.11***

Recall that, as we have mentioned several times, the similarity between cases in our system can not be simply defined as the similarity in appearances (surface features), because this kind of similarity does not guarantee one of two tenets on which *CBR* is based: similar problems have similar solutions (see *§2.1* and *§3.2*). Thus, we have to define our own similarity for the system so that it can retrieve cases that suggest useful hints for the new problem. This similarity is actually defined in the procedure *compareFactors*, which conducts the lower level comparison.

The surface features of factors are in fact used as the main thread by the procedure *compareFactors* to define the similarity between factors, but they are definitely not the only determinants. A large amount of general knowledge about how to determine the similarity between two factors is also implemented into this procedure. Roughly speaking, the factors are compared along with their surface features, such as the type of underlying basic function, the power to which the basic function is raised, whether the factor is on the denominator, whether the factor is under root, *etc.*. But it is the general knowledge which tells the system whether there is a match between two factors being compared along each

surface feature and how good the match is. These knowledge are abstracted from experiences of experienced human problem solvers. They are in the form of rules. Some of the knowledge tell the system when there should (or should not) be a match between two factors. For example, one of the rules states that a factor involving *sin* and factor involving *cos* should be considered to match, even though they appear differently. This is because these two types of functions are often integrated using similar techniques, and the mathematical formulas related to these types of functions often appear to be sort of "symmetric". Another rule states that a factor involving $(x^2 - a^2)$ and a factor involving $(x^2 + a^2)$ should not be considered to match, because methods used to evaluate these two types of functions are quite different (note that the former can be factored, but the later cannot). Some other rules help the system to determine how good a match is (if there is one). For example, one such rule states that, for factors $sin^n x$ and $sin^m x$, if $n = m$ then the match is a perfect one; else if $n$ and $m$ are both even or both odd then the match is a moderate one; otherwise the match is a poor one. This is because trigonometric factors of even powers are often evaluated using similar techniques, and the same could be said to those of odd powers as well. But the technique used for trigonometric factors of even powers is quite different from that for those of odd powers.

The degree of match is measured in terms of nonnegative integers in our system. It consists of two parts. One is the *base-point*, the other is the *bonus-point*. The former is to indicate if there is a match between two factors under consideration, while the later is to indicate how good the match is (if there is one). A nonzero base-point is assigned when the system thinks there is a match between two factors. This is mainly determined by the types of underlining basic functions. Therefore, this step is only a brief preliminary comparison. On the other hand, a bonus-point is considered only when the base-point is nonzero. When the system finds there is a match between two factors, it will go further to compare two

factors along each surface feature to see how good the match is and determine the bonus-point to be assigned, otherwise, it will skip the current pair and continue with the next. This step of comparison is more comprehensive and thorough.

One of the desired (or intentional) "side-effect" of this two-folded comparison schema is that it virtually partitions our case base into sections, so that comparisons will be conducted only in the relevant sections in a search session. This significantly improves the performance of the system.

# Chapter Four

# Conclusions and Discussions

In this chapter, we make some concluding remarks on our system and discuss some of the advantages and disadvantages of the system comparing to the other similar systems. We also address some of the aspects that need future improvement.

***System Limitation***. At this experimental stage, our system supports only three types of factors: *trigonometric*, *quadratic* and *polynomial* factors. By this we mean that, currently, the system takes only these three type of factors as input, and its matching module only knows how to compare these three types of factors. Also, cases in the initial case base (which consists of just over two hundreds cases) contain no other types of factors than these three.

***Testing***. To test the system, we conducted two relatively comprehensive experiments, and the testing results are very encouraging. The first experiment consists of 50 testing samples, which were all randomly chosen from our case base. This experiment was conducted by a person who was not exposed to our indexing schema (but was given a few general instructions on how to determine factors). It turns out that our system retrieved 46 out of the 50 samples (or 92%) from the case base. Analysis showed that the failure in retrieving the remaining 4 samples from the case base was actually caused by the

differences in classifying certain factors by the tester and the system. A simple remedy for this problem will be discussed later in this chapter. In the second experiment, we tested another 50 samples chosen from one of the most popular *Calculus* textbooks by *James Stewart* of the McMaster University. The only criteria for sample selection is that the integrand should contain no other types of factors than those three supported by our system. This experiment was conducted by myself and another person who has a sound mathematical background and years of teaching experiences. The testing result is as following. Among the 50 samples, 23 of them (or 46%) were actually retrieved by the system from its case base. For another 18 of them (or 36%), the system found helpful cases from its case base. Here by helpful cases, we mean the retrieved cases whose solution method can be applied to solve the corresponding sample problem either directly or with some trivial variation (such as substitution $y = x^2$ being replaced by $y = x^3$, or an identity about *sin* being replaced by its counterpart on *cos*, *etc.*). So, altogether, our system provided useful hint for 41 out of 50 testing samples (or 82%). As new cases are added to the case base, this percentage can be definitely improved.

***Case-Based vs. Rule-Based.*** Of the best of my knowledge, this system is the first attempt to apply the *CBR* paradigm in the domain of mathematics. Mathematics, especially in the elementary level, is actually a domain where rules dominate. In other words, most of elementary level mathematical problems (such as those encountered in the *Calculus* course) can be solved based on rules. Before this system, almost all the intelligent systems that we know in this domain (such as *Maple* and *Mathematica*) are rule-based. They use rules to deduce solutions.

Mathematical rules are usually formatted in such a way that makes them as general as possible in order to cover as many situations as possible. These rules are very powerful and reliable, correct answers are almost guaranteed if one can follow the rules. But, there are some disadvantages with the rules as well. For example, for a human being, it is often too

difficult to follow the rules with hard problems, and it is often too tedious to follow the rules with simple ones. Anyone with some mathematical training may have the experience that, for a particular problem, "tricks" often do more help than general methods (rules). By "tricks", we mean those special skills for particular problem(s). This is because those tricks take into account the special properties of the particular problem(s). They deal with the problem(s) more directly, therefore they often lead to faster and easier solutions. And, most important of all, they are easier to be carried out by human.

The most significant difference between our system and other *CAS*s (*computer algebraic systems*) can be summarized as following. Those *CAS*s implement general rules so that they can actually *solve* problems and produce correct answers (although the solutions may not be nice). Therefore, for people who need only answers to problems but do not care much about how problems are solved (for example, some engineers and/or researchers in certain areas), those *CAS*s are very good choices. On the other hand, our system do not actually "solve" any problem. Instead, it stores "tricks" (special skills, experiences, or knowledge) which can be used to solve problems and makes them available when needed. These tricks are very specific and easy to be carried out by human. It is fair to say that our system teaches the problem-solving methods rather than solving problem for users. Therefore, our system is ideal for people who are more interested in knowing how a particular integration problem can be solved than just the answer, for example, students who take the *Calculus* course.

***Accessibility.*** Another unique characteristic of our system is, that it is accessible through the Internet. We fully take the advantages of the vigorous internet technology to make our system reachable to anyone anywhere anytime. This feature makes our system a more conveniently accessible agent than human agents (such as instructors, TAs, tutors *etc.*). With this system in the Internet, students having trouble in integration problems do not have to wait till their instructors' next office hour and do not have to wait in the long lineup outside their instructor's office for helps.

***Learning Prospect.*** At this stage, our system may not be as intelligent as those very experienced human agents. But, it does have the ability to learn new skills and new experiences from human agents. As more and more new cases are added to its case base in the future, its expertise may increase and cover a wider range of problems.

Now, we discuss some of the aspects of the system that need further improvements and can be further improved.

***Scale-up Problem.*** As we mentioned, our system currently supports only three type of factors. For it to be more practically usable, the system need to be scaled-up to support all sorts of factors, such as logarithmic factors, exponential factors, hyperbolic factors, and so on. This is definitely possible and achievable, but need quite a large amount of effort. For this purpose, a large number of new cases need to be added to the case base, which should cover not only those functions involving these types of factors but also the combinations and/or compositions of these types of factors with those existing types of factors. Also, the algorithm for comparing those types of factors need to be implemented in the matching module as well.

***Multiple Indexing.*** Currently, in our system, each case has exactly one index entry in the index file. Notice that some factors can be categorized into different types by different persons. For example, some one may classify $(x^2 + 1)$ as a polynomial factor while others may classify it as a quadratic factor. They are both perfectly correct. But, if users classification does not match the system's internal classification, than the system may not be able to pick it up even though the function is in the case base. A simple remedy to this problem is to keep multiple indexes in the index file for one case if necessary. No other change is needed for this remedy. Currently multiple indexing can only be done manually. We do need a utility to automate this process for the long run.

***Answers: Graphical Expression vs. Textual Expression.*** Currently answers to our

cases are expressed and stored in textual format using *Maple* syntax. Solutions to some cases are so lengthy and/or complex that their textual expressions may be difficult to understand or even confusing for students who are not familiar with the *Maple*. One of the obvious remedy for this problem is to use graphical expressions. It is not difficult to make this change at all, if we have enough time to generate hundreds of images and enough space to store them in our server.

# Appendix A

# Format of the Index String Variables

In order to properly index the mathematical functions so that the system can store and retrieve them, we classify factors into categories according to the types of underlying basic functions. For each category, we assign a formatted string variable which is used to describe the factors in the category. Each byte in a string variable takes special values and has particular meaning. Currently, the system supports only three types of factors (see *§3.2*): *trigonometric*, *quadratic* and *polynomial* factors. In the following, we present the details about the format of the string variables for these three types of factors.

*1.* ***String variable for trigonometric factors*** is formatted as following:

```
trig xxx xxxxxx x x x x x x
     ①   ②    ③ ④ ⑤ ⑥ ⑦ ⑧
```

① This 3-byte segment indicates the type of the underlying trigonometric function. It takes following values: *sin, cos, tan, cot, sum*. The last value *sum* is used to accommodate the factors containing simple sum of trig functions such as (*sin x* + *cos x*) or (*tan x* - *cot x*).

② This 6-byte segment is used to describe what types of trig functions are involved in the sum when the value for ① is *sum*. Each byte takes either 1 or 0 to indicate the presence

of certain type of basic function. The assignment is as following: (*sin cos tan cot x a*). We allow x and a (for constants) to accommodate factors such as ($x + sin\ x$) or ($1 + cos\ x$).

③ This byte is used to record the power to which the underlying basic function is raised. The values it takes are: 1, 2, ..., 9, n. We do not discriminate among powers higher than 9. The default value is 1.

④ This byte is used to indicate if the underlying basic function is in the denominator. The values it takes are either y or n. The default value is y.

⑤ This byte is used to indicate if the underlying basic function is under root. The values it takes are either y or n. The default value is y.

⑥ This byte is used to record the order of the root if ⑤ has value y. The values it takes are: 1, 2, ..., 9, n. We do not discriminate orders higher than 9. The default value is 1.

⑦ This byte is used to record the power to which the variable (usually $x$) in the underlying basic function is raised. It is to accommodate factors such as $sin(x^2)$. The values this byte takes are: 1, 2, ..., 9, n. We do not discriminate among powers higher than 9. The default value is 1.

⑧ This byte is used to record the coefficient of the variable in the underlying basic function. The values it takes are: 1, 2, a, -1, -2, -a. We do not discriminate coefficients larger than 2 or less than -2. The default value is 1.

***2. String variable for quadratic factors*** is formatted as following:

```
quad xxx x x x x x
     ① ② ③ ④ ⑤ ⑥
```

① This 3-byte segment indicates the type of the underlying quadratic functions. It takes following values: x+a, x-a, a-x, and abc, which represent functions of the forms $(x^2+a^2)$, $(x^2-a^2)$, $(a^2-x^2)$, and $(ax^2+bx+c)$, respectively.

② This byte is used to record the power to which the underlying basic function is raised. The values it takes are: 1, 2, ..., 9, n. We do not discriminate among powers higher than 9. The default value is 1.

③ This byte is used to indicate if the underlying basic function is in the denominator. The values it takes are either y or n. The default value is y.

④ This byte is used to indicate if the underlying basic function is under a root. The values it takes are either y or n. The default value is y.

⑤ This byte is used to record the order of the root if ⑤ has value y. The values it takes are: 1, 2, ..., 9, n. We do not discriminate among orders higher than 9. The default value is 1.

⑥ This byte is used to record the coefficient of the variable in the underlying basic function. The values it takes are: 1, a, -1, -a. We do not discriminate among coefficients larger than 1 or less than -1. The default value is 1.

*3. **String variable for polynomial factors*** is formatted as following:

```
poly xxx x x x x x x
     ① ② ③ ④ ⑤ ⑥ ⑦
```

① This 3-byte segment indicates which function the underlying polynomial functions

is in term of. It takes following values: _x_, sin, cos, tan and cot. This is designed to accommodate more complicated functions such as $(sin^2x + sinx + 1)$. The default value is _x_ (representing polynomial in term of x).

② This byte is used to record the power to which the underlying basic function is raised. The values it takes are: 1, 2, ..., 9, n. We do not discriminate among powers higher than 9. The default value is 1.

③ This byte is used to indicate if the underlying basic function is in the denominator. The values it takes are either y or n. The default value is y.

④ This byte is used to indicate if the underlying basic function is under root. The values it takes are either y or n. The default value is y.

⑤ This byte is used to record the order of the root if ⑤ has value y. The values it takes are: 1, 2, ..., 9, n. We do not discriminate orders higher than 9. The default value is 1.

⑥ This byte is used to record the highest power the selected function in ① is raised to. The values it takes are: 1, 2, ..., 9, n. We do not discriminate among powers higher than 9. The default value is 1.

⑦ This byte is used to record the coefficient of the variable in the underlying basic function. The values it takes are: 1, a, -1, -a. We do not discriminate among coefficients larger than 1 or less than -1. The default value is 1.

# References


[1] **Aamodt**, A. (1990). Knowledge-intensive case-based reasoning and learning. ECAI-90, Ninth European Conference on Artificial Intelligence, Stockholm, August.

[2] **Aamodt**, A. (1991). Problem Solving and Learning from Experience. An Introduction to Case-Based Reasoning. NAIM - Nordic AI Magazine, Volume 6, no. 1, pp.19-25.

[3] **Aamodt**, A. (1993). A case-based answer to some problems of knowledge-based systems. In E. Sandewal and C.G. Jansson (eds): Scandinavian Conference on Artificial Intelligence 1993. IOS Press, pp. 168-182.

[4] **Aamodt**, A. (1994). Explanation-driven case-based reasoning, In S. Wess, K. Althoff, M. Richter (eds.): Topics in Case-based reasoning. Springer-Verlag, pp 274-288.

[5] **Aamodt**, A. & **Plaza**, E. (1994). Case-based reasoning; Foundational issues, methodological variations, and system approaches. AI Communications, Vol.7, No.1, March , pp. 39-59.

[6] **Acorn**, T. & **Walden**, S. (1992). SMART: Support management cultivated reasoning technology for Compaq customer service. In Proceedings of AAAI-92. Cambridge, MA: AAAI Press/MIT Press.

[7] **Aha**, D.W. (1991). Case-based learning algorithms. In Proceedings of the DARPA Case-Based Reasoning Workshop (pp. 147-158). Washington, D.C.: Morgan Kaufmann.

[8] **Aha**, D.W., & **Goldstone**, R.L. (1990). Learning attribute relevance in context in instance-based learning algorithms. In Proceedings of the Twelfth Annual Conference of the Cognitive Science Society (pp. 141-148). Cambridge, MA: Lawrence Erlbaum.

[9] **Albert**, M.K., & **Aha**, D.W. (1991). Analysis of instance-based learning algorithms. In

Proceedings of the Ninth National Conference on Artificial Intelligence (pp. 553-558). Anaheim, CA: AAAI Press.

[10] **Allen**, B.P. (1994). Case-Based Reasoning: Business Applications. Communications of the ACM, Vol. 37, pp 40-42, March.

[11] **Althoff**, K.D. (1992). Machine learning and knowledge acquisition in a computational architecture for fault diagnosis in engineering systems. Proceedings of the ML-92 Workshop on Computational Architectures for Machine Learning and Knowledge Acquisition. Aberdeen, Scotland, July.

[12] **Ashley**, K.D. & **Rissland**, E.L. (1988). A Case-Based Approach to Modeling Legal Expertise. IEEE Expert, Vol. 3, No. 3, pp. 70-77.

[13] **Auriol**, E.; **Wess**, S.; **Manago**, M.; **Althoff**, K.D. & **Trahöner**, R. (1995). INRECA: A seamlessly integrated system based on inductive inference and case-based reasoning. In Proceeding of First International Conference on Case-Based Reasoning, pp 371-380. Berlin: Springer-Verlag.

[14] **Avesani**, P., **Perini**, A. & **Ricci**, F. (1993). Combining CBR and Constraint Reasoning in Planning Forest Fire Fighting. In Proceedings of 1st European Workshop on Case-Based Reasoning, Kaiserslautern, November 1-5.

[15] **Bareiss**, R. (1988). ***PROTOS***: a unified approach to concept representation, classification and learning. Ph.D. Dissertation, University of Texas at Austin, Dept. Of Computer Science 1988. Technical Report AI88-83.

[16] **Barletta**, R. (1994). A hybrid indexing and retrieval strategy for advisory CBR systems built with ReMind. In Proceedings of the Second European Workshop on Case-Based Reasoning, pp. 49-58.

[17] **Berger**, J. (1994). ROENTGEN: Radiation Therapy and Case-Based reasoning. Proceedings of the 10th Conference on AI for Applications, IEEE Computer Society Press.

[18] **Branting**, L.K., & **Aha**, D.W. (1995). Stratified case-based reasoning: Reusing hierarchical problem solving episodes. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (pp. 384-390). Montreal, Canada: Morgan Kaufmann.

[19] **Branting**, L.K. and **Aha**, D.W., (1995). Stratified Case-Based Reasoning: Reusing

Hierarchical Problem Solving Episodes. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), pp. 384-390, Montreal.

[20] **Burstein**, M. (1988). Combining analogies in mental models. In Helman, D. (Ed.): Analogical Reasoning. Boston, Kluwer.

[21] **Callan**, J.P., **Croft**, W.B. & **Harding**, A.M. (1992). The INQUERY Retrieval System. In Tjoa A.M. & Ramos, I. (Eds.): Database and Expert Systems Applications: Proceedings of the International Conference in Valencia, Spain, pp 78-83. Springer-Verlag.

[22] **Chandler**, T.N. (1994). A Case-Based Advising System for Lesson Planning.

[23] **Chandler**, T.N. and **Kolodner**, J. (1993). The Science Education Advisor: A Case-Based Advising System for Lesson Planning. Proceedings of the International Conference on AI and Education, Scotland, August.

[24] **Domeshek**, E.A. & **Kolodner**, J.L. (1992). A Case-Based Design Aid for Architecture. Proceedings: AI and Design Conference, June, pp.497-516.

[25] **Domeshek**, E.A., **Kolodner**, J.L. & **Zimring**, C.M. (1994). The design of a toolkit for case-based design aids. Proceedings: AI and Design Conference, May, pp.109-126.

[26] **Domeshek**, E.A.; **Herndon**, M.D.; **Bennett**, A.W. & **Kolodner**, J.L. (1994). A Case-based design aid for conceptual design of aircraft systems. Proceedings of the 1994 IEEE Conference in AI Applications.

[27] **Ellman**, J. (1995). An application of case-based reasoning to object-oriented database retrieval. In Watson, I. (Ed.): Progress in Cased-Based Reasoning: Fist United Kingdom Workshop, pp 134-141. Springer-Verlag.

[28] **Faries**, J. & **Schlossberg**, K. (1994). The effect of similarity on memory for prior problems. In Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society, Hillsdale, NJ. Lawrence Erlbaum. pp. 278-282.

[29] **Francis**, A.G. & **Ram**, A. (1993). Computational Models of the Utility Problem and their Application to a Utility Analysis of Case-Based Reasoning. In ML-93 Workshop on Knowledge Compilation and Speedup Learning, Amherst, MA, June.

[30] **Francis**, A.G. & **Ram**, A (1993). The Utility Problem in Case-Based Reasoning. In Abstracted in the AAAI-93 Workshop on Case-Based Reasoning, Washington, DC, July.

[31] **Goodman**, M. (1990). Prism: A case-based telex classifier. In Rappaport, A. & Smith, R. (Eds.): Innovative Applications of Artificial Intelligence, Vol.2, Cambridge, MA, MIT Press.

[32] **Hammond**, K.J. (1990). Case-Based Planning: A Framework for Planning Experience. The J. of Cognitive Science, Ablex Publishing, Norwood, NJ, Vol. 14, No.13, September.

[33] **Hammond**, K.J.; **Burke**, R. & **Schmitt**, K. (1996). A Case-Based Approach to knowledge Navigation. Working Report of the AAAI Workshop on Multi-Media and artificial Intelligence, July.

[34] **Hennessy**, D.H. & **Hinkle**, D. (1992). Applying cased-based reasoning to autoclave loading. IEEE Expert 7(5), pp 21-26.

[35] **Hunter**, L. (1989). Gaining expertise through experience. Ph.D. dissertation, Computer Science Department Technical Report #678, Yale University.

[36] **Jurisica**, I. & **Glasgow**, J. (1995). Applying Case-Based Reasoning to Control in Robotics. 1995 Robotics and Knowledge-Based Systems Workshop, St. Hubert, Quebec.

[37] **Klein**, G. & **Calderwood**, R. (1988). How do people use analogues to make decisions? In Kolodner, J.L. (Ed.): Proceedings of the DARPA Case-Based Reasoning Workshop, pp. 209-223. Palo Alto: DARPA.

[38] **Klein**, G. & **Calderwood**, R. (1989). Decision models: Some lessons from the field. IEEE Transactions on Systems, Man, and Cybernetics, 21(5), pp. 1018-1026.

[39] **Kolodner**, J.L. (1983) Maintaining organization in a dynamic long-term memory. Cognitive Science, Vol. 7, pp 243-280.

[40] **Kolodner**, J.L. (1991). Improving Human Decision-Making Through Case-Based Decision Aiding. The AI Magazine, 12 (2), pp. 52-68.

[41] **Kolodner**, J.L. (1993). Case-based Reasoning, Morgan Kaufmann Publishers, San Mateo, CA.

[42] **Kolodner**, J.L. (1994). From natural language understanding to case-based reasoning and beyond: A perspective on the cognitive model that ties it all together. In Langer, E. and Schank, R.C. (Eds.), Reasoning and Decision Making: Psycho-Logic in Honor of Bob Abelson, Lawrence Erlbaum Associates Inc., Northvale, NJ.

[43] **Koton**, P. (1988). Integrating case-based and causal reasoning. In Proceedings of the Tenth Annual Conference of the Cognitive Science Society. Northvale, NJ. Erlbaum.

[44] **Koton**, P. (1989). Using experience in learning and problem solving. Massachusetts Institute of Technology, Laboratory of Computer Science (Ph.D diss., 1988). MIT/LCS/TR-441.

[45] **Lancaster**, J.S. & **Kolodner**, J.L. (1987). Problem solving in a natural task as a function of experience. In: Proceedings of the Ninth Annual Conference of the Cognitive Science Society. Nothvale. NJ. Erlbaum.

[46] **Lancaster**, J.S. & **Kolodner**, J.L. (1988).Varieties of learning from problem solving experience. In: Proceedings of the Tenth Annual Conference of the Cognitive Science Society. Nothvale, NJ. Erlbaum.

[47] **Leake**, D.B. (1996). CBR in context: The present and future. In Leake, D. (Ed.): Case-Based Reasoning: Experiences, Lessons, and Future Directions. Menlo Park: AAAI Press.

[48] **Lebowitz**, M. (1983). Memory-based parsing. Artificial Intelligence 21, pp. 363-404.

[49] **López**, B. & **Plaza**, E. (1990). Case-based learning of strategic knowledge. In Kodratoff, Y. (Ed.): Machine Learning-EWSML-91, Lecture Notes in Computer Science, pp. 389-411 Springer-Verlag).

[50] **Maguire**, P.; **Szegfue**, R.; **Shankararaman**, V. & **Morss**, L. (1995). Application of case-based resoning (CBR) to software reuse. In Watson(Ed.): Progress in Cased Reasoning: Fist United Kingdom Workshop, pp 166-174. Springer-Verlag,.

[51] **Mark**, W. (1989). Case-based reasoning for autoclave management. In Hammond, K. (Ed.): Proceedings for Workshop on Case-based Reasoning (DARPA). Pensacola Beach, Florida. Morgan Kaufmann.

[52] **Moorman**, K. & **Ram**, A. (1992). A Case-Based Approach to Reactive Control for Autonomous Robots. In AAAI Fall Symposium on AI for Real-World Autonomous Robots, Cambridge, MA, October.

[53] **Pirolli**, P. & **Anderson**, J. (1985).The role of learning from examples in the acquisition of recursive programming skills. Canadian Journal of Psychology, 39, pp. 240-272.

[54] **Porter**, B. & **Bareiss**, R. (1986). PROTOS: An experiment in knowledge acquisition for heuristic classification tasks. In: Proceedings of the First International Meeting on

Advances in Learning (IMAL), Les Arca, France, pp 159-174.

[55] **Ram**, A. and **Santamatia** I.C., (1993). Continuous Case-based Reasoning. Proceedings of the AAAI-93 Workshop on Case-Based Reasoning, pp 86-93, Washington, DC. July.

[56] **Ram**, A. & **Santamaria**, J.C. (1993). A Multistrategy Case-Based and Reinforcement Learning Approach to Self-Improving Reactive Control Systems for Autonomous Robotic Navigation. In Second International Workshop on Multistrategy Learning, Harpers Ferry, WV, May.

[57] **Read**, S. & **Cesa**, I. (1991). This reminds me of the time when ...: Expectation failures in reminding and explanation. Journal of Experimental Social Psychology, 27, pp. 1-25.

[58] **Reategui**, E.B. & **Campbell**, J.A. (1994). A classification system for credit card transactions. In Haton, J.-P., Keane, M. & Manago, M. (Eds.): Advances in Case-Based Reasoning, Selected Papers from the Second European Workshop on Case-Based Reasoning. November, Springer-Verlag, pp 165-177.

[59] **Richter**, A.M. & **Weiss**, S. (1991). Similarity, uncertainty and case-based reasoning in PATDEX. In Boyer, R.S. (Ed.): Automated reasoning, essays in honor of Woody Bledsoe. Kluwer, pp. 229-265.

[60] **Riesbeck**, C. (1988). An interface for case-based knowledge acquisition. In Kolodner, J.L. (Ed.): Proceedings of the DARPA Case-Based Reasoning Workshop, pp. 312-326. San Mateo, Morgan Kaufmann.

[61] **Rissland**, E.L.; **Daniels**, J.J.; **Rubinstein**, Z.B. & **Skalak**, D.B. (1993). Case-Based Diagnostic Analysis in a Blackboard Architecture. Proceedings of the Eleventh National Conference on Artificial Intelligence, 66-72. Washington, DC. AAAI Press/MIT Press.

[62] **Rissland**, E.L & **Daniels**, J.J. (1995). Using CBR to Drive IR. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), pp 400-407, Montreal.

[63] **Ross**, B.H. (1984). Remindings and their effects in learning a cognitive skill. Cognitive Psychology, 16, pp. 371-416.

[64] **Ross**, B.H. (1986). Remindings in learning: Objects and tools. In Vosniadou, S. & Ortony, A. (Eds.): Similarity, analogy, and thought. Cambridge University Press, New York.

[65] **Ross**, B.H. (1989). Some psychological results on case-based problem solving. In Hammond, K. (Ed.): Proceedings for Workshop on Case-based Reasoning (DARPA). Pensacola Beach, Florida. Morgan Kaufmann.

[66] **Schank**, R. (1982) Dynamic memory: A theory of learning in computers and people. New York: Cambridge University Press.

[67] **Schmidt**, H.; **Norman**, G. & **Boshuizen**, H. (1990). A cognitive perspective on medical expertise: Theory and implications. Academic Medicine 65(10), pp. 611-621.

[68] **Sycara**, K. (1987). Resolving adversarial conflicts: An approach to integrating case-based and analytic methods. Georgia Institute of Technology, School of Computer Science Technical Report No. GIT-ICS-87/26. Atlanta.

# VITA AUCTORIS

| | |
|---|---|
| NAME: | Qin Xu |
| PLACE OF BIRTH: | Shenyang, Liaoning, China |
| YEAR OF BIRTH: | 1960 |
| EDUCATION: | University of Windsor, Windsor, Ontario, Canada<br>M.Sc., Computer Science, 1997 |
| | University of Alberta, Edmonton, Alberta, Canada<br>Ph.D., Mathematics, 1993 |
| | Northeastern University, Shenyang, Liaoning, China<br>M.Sc., Mathematics, 1985 |