Chapter 1

INTRODUCTION AND OVERVIEW

1.1 Introduction

Whatever intelligence may be, or be denned as, reasoning and problem-solving have traditionally been viewed as central subsets of it. Reasoning is the art of finding out what information follows from what other information, of finding what information is consistent with what other information, of finding what information is needed to answer a problem, and how to derive that answer. An important characteristic of reasoning is the combination of information items to form new information, usually in the process of deriving a particular conclusion by carefully considering the available facts.

Logic plays an important role in reasoning and problem-solving. It offers a formal mechanism for learning. The power of logic is based on three important features: Firstly, it provides a language for the accurate representation of knowledge. Secondly, a framework for processing the represented knowledge is given in the form of a calculus defining permitted rules for drawing conclusions. Thirdly, a mechanism for mechanical proofs of truth values, or equivalence, of statements can be defined [Kurff 89]. Prepositional logic is appealing because it is simple to deal with and a decision procedure for it exists. Real-world facts can be represented as logical propositions written as well-formed formulas (wffs) in prepositional logic. The molecules of logic are statements - and statements are chunks of knowledge or information.
1.1.1 Search methods

Search problems are ubiquitous in AI. Almost every AI program depends on a search procedure to perform its prescribed function. For every problem encountered there might be numerous alternatives to consider. The problem-solvers are constantly confronted with the necessity to select among these equally plausible alternatives. The frustrating property exhibited by most of the search problems is the exponential growth of the plausibilities. The computing time for the problem is hard to control as it grows exponentially due to combinatorial explosion. Though search is a general mechanism for intelligence, the efficiency with which it can be performed limits its applicability. The central issue in search is efficiency. Two important measures of efficiency are the amount of time and the amount of memory required to find the solution or to conclude absence of a solution.

Brute-force technique

Brute-force method of search guarantees a solution if there is one. This is a blind search in the sense that it uses no knowledge about the problem other than the problem space itself. All the alternatives in the search space are explored mechanically and tested until a solution is found (or all solutions are found), a time limit has been reached or failure occurs. At the worst case, it may be necessary to explore the whole search space before finding a solution.

If each point of the search space is dissimilar, Brute-force method is the best that can be applied. However, in most cases, there is a great deal of similarity among the points of the search space. Hence, in order to achieve efficiency, the result obtained in one region of the search space has to be carried to other regions.
ability to perform dependency directed backtracking and so to support non-monotonic reasoning. TMS are also called belief revision or Reason Maintenance Systems (RMS). In this thesis the terms RMS and TMS are used interchangeably.

A RMS is the house-keeping subsystem of an overall reasoning system. The basic architectural presupposition is that the overall reasoning system consists of two components: a problem-solver and a TMS [de Kleer 86a]. The problem-solver usually includes all domain knowledge and inference procedures. Every inference made by the problem-solver is communicated to the TMS. An important characteristic of RMS is that it has no access to the semantics of the problem-solver behaviour. The RMS treats the expressions passed to it by the problem-solver purely syntactically. As a consequence of this, the problem-solver is held responsible for the correctness of the information passed to the RMS.

The uniqueness of RMS stems from maintaining records of the origins of labels assigned to database facts (dependency records), and subsequently using those dependency records to prune the search space and perform database updating. Different searching and reasoning programs had used many of the ideas previously, in a somewhat adhoc way. Hayes [Hayes 75] seems to be the earliest reference to what might be regarded as a RMS. However, Doyle [Doyle 78, Doyle 79] provided the first non-monotonic comprehensive implemented version of a TMS which automatically maintains consistency.

A context is normally determined by some set of hypotheses or assumptions and is expected to be consistent. Based on the type of dependency storage the systems are referred to as justification based or assumption based. Based on the type of access to information they are referred to as single context or multiple context systems.

The RMS insists on maintaining for the problem-solver a single context that has been passed to the RMS in a single context system. The multiple context system provides a facility for determining contexts dynamically, without enforcing the usage of any particular
one. Typically, justification based systems operate with single context and assumption based systems operate with multiple contexts.

Justification-based TMS (JTMS)

The JTMS associates a special data structure, called a node, with each problem-solver datum (formula in the database). These nodes are connected together in a web of data dependencies. The TMS uses Horn clauses as justifications. Each node has a status in or out, and a justification. If its justification is valid, then a node is in, and otherwise it is out. In Doyle's TMS [Doyle 79], a justification has the form (<inlist><outlist>) and is valid if all the nodes in its inlist are in and all the nodes in the outlist are out. The support for an in node must be well-founded, that is, circular, self-supporting networks of justifications are not valid.

The form of justifications permits non-monotonic inference. In other words, the coming in of a node that was previously out can result in the going out of a node that was previously in. This facilitates the creation of revisable assumptions as well as non-revisable premises. RMS performs two basic operations on the web of dependencies: reason maintenance and dependency directed backtracking. Reason maintenance is invoked whenever the problem-solver adds a new node or justification, and it ascertains which nodes are in and which are out. Dependency directed backtracking is invoked to resolve contradictions by backtracking through the thread of justifications for the contradictory node in search of an assumption which it can retract in order to restore consistency.

Shortly after Doyle, McAllester [McAllester 80] designed a single context Logic-based TMS (LTMS) which is significantly more efficient and comprehensible than Doyle’s TMS. LTMS labels nodes as TRUE, FALSE or UNKNOWN and uses disjunctive clauses as justifications.
Chapter 1. INTRODUCTION AND OVERVIEW

ATMS as an assumption. The basic structure on which the system depends is the ATMS node. Conceptually, the node has three parts. These are the problem-solver datum, a label representing the assumptions under which the datum holds, and the justifications provided by the problem-solver which supports the datum. A set of assumptions is an environment and the set of all data propositionally derivable from the assumptions using the justifications is the context of the environment. A datum is in a context if it is propositionally derivable (using the justifications) from the assumptions of the context. An environment is inconsistent, if falsity (symbolised as \( \bot \)) is propositionally derivable from the assumption set. An inconsistent environment is defined as not having a context. The efficiency of the ATMS is based on the observation that if a datum is derivable from a particular set of assumptions it is derivable from every superset as well. The ATMS associates with every datum the minimal set of environments from which it is derivable. This set is the label of the datum. By computing the label for each datum, the ATMS indirectly computes the contents of each context. Given the label, it is very easy to compute the contents of each context. A datum is in a context exactly when the assumptions of the context are a subset of any of the environments of the datum's label.

An ATMS offers an improvement over conventional RMS for search problems where all or many solutions are required. The problem-solver can explore many possibilities at once, and can compare solutions and potential solutions to problems. Furthermore, the resulting mechanism obviates the need for backtracking. When only one or a few solutions are required, the conventional RMS is more efficient. Acknowledging this and other deficiencies, de Kleer and Williams [de Kleer 86d] proposed a hybrid algorithm called assumption based dependency directed backtracking. ATMS, for many tasks, is more efficient than previous TMSs and has a more coherent interface between the TMS and the problem-solver without giving up exhaustivity. Unlike previous TMSs which are based on manipulating justifications, the ATMS is, in addition, based on manipulating
Backtracking

*Chronological backtracking* helps to increase the efficiency by reducing some of the futile search. The alternatives are selected following some order from one alternative to another and requires an additional machinery for controlling the search. If at any stage the search element is inconsistent or contradictory, the chronological backtracker retracts to the most recent selection made and proceeds from that point. Though this is better than the brute-force technique, much of the work undertaken by a chronological backtracking search procedure when it encountered a failure might be irrelevant to the particular problem discovered, and some of the backtracking may lead only to rediscover the contradictions. Moreover, if the failure depends only on choices made much earlier, all the work done in remaking the later choices is potentially irrelevant. Rather than retracting to the most recent selection made, the retraction to the selection which is responsible for the inconsistency could be much efficient. Information about the choices on which an inference rests is stored so that the culprits for the failure can be identified and the problem-solver need re-make only those choices. The dependency relations between choices and failure are used to direct the backtracking. This idea of *dependency directed backtracking* originated with Stallman and Sussman [Stallman 77].

1.1.2 Reasoning systems

In conventional reasoning systems, much of the work such as the caching of expensive inferences was implemented anew for each problem encountered. Clearly this is wasteful and, because these support mechanisms were not always separated from the problem-solving, it could lead to unnecessary confusion in the system design [Kelleher 88].

These concerns prompted the need to perform belief revision and motivated the development of *Truth Maintenance Systems* (TMS) [Doyle 79] as a way of providing the
Chapter 1. INTRODUCTION AND OVERVIEW

In order to make recomputing of the status of an assertion and switching of contexts easier, McDermott [McDermott 83] attempted to bring together the ideas of data dependencies (as in Doyle’s TMS) and contexts. The system works with a current context, defined by the user as a set of premises. It computes a label, a Boolean expression in the premises, for each assertion using the current justification set. The value of the label is found by assigning the value true to the premises in the current context. This value of the label determines the current status of the assertion. The status of any assertion in all possible data contexts can be determined using this label. This makes switching to a different context simple. However, McDermott did not exploit this idea further to produce a true multiple context system. McDermott’s system can be seen as overlapping the boundary between justification based systems and assumption based systems. The RMSs based on assumptions, appear to have started with the work of Martins and Shapiro [Martins 83, Martins 88], but the most widely used example of the assumption based approach to reason maintenance is probably de Kleer’s [de Kleer 86a].

Assumption based TMS

In 1986, de Kleer introduced the idea of an Assumption-based Reason Maintenance System (ATMS) which solve several inherent problems in earlier TMS implementations. In Doyle’s TMS, only one labelling of nodes with in or out is considered at any one time. The problem-solver can only focus on a single set of assumptions and their consequences. In contrast to this, the ATMS incrementally computes the assumptions on which each datum depends as each new problem-solver inference is received. The mechanisms of the ATMS revolve around its ability to determine, for any given item of data, the assumptions under which it holds.

All inferences made by the problem-solver are recorded and communicated to the ATMS as justifications and every problem-solving hypothesis is communicated to the
In a justification-based TMS, the database is always kept consistent; this makes it impossible to refer to problem-solving contexts explicitly and requires truth maintenance and dependency-directed backtracking to move to a different point in search space. On the other hand, in an ATMS each datum is labelled with the sets of assumptions (representing the contexts) under which it holds. These sets of assumptions are computed by the ATMS from the problem-solver-supplied justifications. The idea is that the assumptions are the primitive data from which all the other data are derived. These assumption sets can be manipulated far more conveniently than the datum sets they present. There is no necessity that the overall database be consistent; it is easy to refer to contexts, and moving to a different point in the search space requires very little work.

Conventional TMS is oriented towards finding one solution whereas ATMS is oriented towards problem-solving in multiple contexts simultaneously. This is efficiently achieved by labelling each datum with the assumptions upon which it ultimately depends. This idea and its ramifications radically alters the conception and technology of problem-solving. ATMS is not a panacea and is not suited to all tasks.

The basic ATMS [de Kleer 86a] provides a novel truth maintenance facility. Reiter and de Kleer [Reiter 87] proposed a generalization of the basic ATMS called the Clause Management System (CMS) and showed its applications to abductive reasoning. A CMS is intended to work together with a reasoner, which issues queries that take the form of clauses. The CMS is then responsible for finding minimal supports for the queries. Reiter and de Kleer [Reiter 87] show some relationships between prime implicants and minimal supports.
An ATMS is precisely intended to generate all and only minimal explanations simultaneously [Inoue 89], given a set of clauses. In the ATMS terminology, the set of minimal explanation of a node from the justifications and the assumptions is called the label of node, which is consistent, sound, complete and minimal. The basic ATMS is restricted to accept only Horn clause justification and atomic assumptions. If justification can contain non-Horn clauses, and the assumptions are allowed to be literals, then this generalization covers de Kleer's various extended versions of ATMS [de Kleer 86a, de Kleer 86b, de Kleer 86c], Dressler's extended ATMS [Dressier 90], and Reiter and de Kleer's Clause maintenance System (CMS) [Reiter 87].

There have been different algorithms to compute the minimal supports. However, Reiter and de Kleer [Reiter 87] consider two ways in which the CMS manages the knowledge base: keeping the set of clauses (denoted by $\mathcal{F}$) transmitted by the reasoner as it is (the interpreted approach), or computing the prime implicants/implicates of $\mathcal{F}$ (the compiled approach). When we are faced with a situation where we want to know explanations for many different queries, we must run the algorithm each time a query is issued. Instead of keeping the initial formula $\mathcal{F}$ as it is and doing the same deductions over and over again for different query clauses, some of these inferences can be made once and for all. That is the motivation for the compiled approach. One of the disadvantages of the compiled approach is the high cost of updating the knowledge base. When the reasoner adds a clause $D$ to $\mathcal{F}$, we must compute all prime implicates of $\mathcal{F} A D$. There are many approaches to compute the prime implicants/implicates. These methods are discussed in Section 1.3.
1.2 Hypothetical Reasoning

The information/knowledge available can be imperfect in one or more respects in the sense that it can be uncertain, incomplete, imprecise, inconsistent, or a combination of these. Most real world reasoning is performed in the context of imperfect information. Processing of imperfect information plays an important role in realizing advanced AI functions such as common sense, learning, automated reasoning etc. [Poole 87, Poole 88, Ishizuka 90]. A non-monotonic reasoning system is required to handle incomplete knowledge in the knowledge-base. Its formalism has a close connection with constraint satisfaction problem.

There are many approaches to the study of imperfect information processing especially when the knowledge is incomplete. Hypothetical reasoning is one of the reasoning schemes which handles incomplete knowledge as hypotheses. The central function of hypothetical reasoning is abductive inference which generates necessary combinations of hypotheses for proving a given goal. The hypothetical reasoning system is a logic-based one, where the knowledge is divided into two categories, i.e. complete knowledge $\mathcal{F}$ and set $\mathcal{H}$ of hypotheses. Complete knowledge is always true and has no possibility of inconsistency. On the other hand, set of hypotheses is incomplete, or defeasible knowledge, for which consistency checking is required in the inference process. The basic behaviour of the hypothetical reasoning [Ishizuka 91] is as follows. When a goal (or an observation) is given the system tries to prove the goal from the complete knowledge. If it fails, then the system selects a subset of the hypotheses so that the given goal is proved from the union of complete knowledge with this subset. The selected subset of the hypotheses should be consistent with complete knowledge, while inconsistency is allowed in the whole set of hypotheses. To efficiently exclude inconsistent combinations of hypotheses, truth maintenance is necessary in this inference process. A reasoning system based on logic
(logic-based reasoning system) can deal with incomplete knowledge as hypotheses. It is a useful framework because of its theoretical basis and applicability. In ordinary logic-based problem-solving, the success or failure of deductive proof becomes the answer. When the goal includes variables, the binding (unification) to the variables becomes an answer in the success case. On the other hand, a selected subset of the hypotheses becomes an answer in the logic-based hypothetical reasoning system, in which deductive inference mechanism is utilized in reverse direction to generate a solution hypotheses subset.

While it is a useful knowledge-processing framework applicable to many practical problems, the most crucial problem of logic-based hypothetical reasoning system is its slow inference speed. An immediate remedy for this problem is to incorporate heuristic knowledge which plays the role of guiding the inference. However, it is difficult to cover the whole problem domain by heuristic knowledge, which causes the well-known knowledge acquisition problem. Therefore, a fast inferencing mechanism not relying on heuristic knowledge is required. Backtracking caused by the inconsistency among selected hypotheses is the major factor of deteriorating the inference speed. A two-phase hypothetical reasoning system has been presented by Ishizuka [Ishizuka 91] where a goal-directed inference-path network is formed using the complete knowledge set but excluding hypotheses in the first phase. Hypotheses necessary for proving a given goal are synthesized in the second phase along this inference-path network in a forward inference fashion with no backtracking. The inference-path network also allows to reduce the number of computationally expensive hypotheses combination to a minimum. The formation of the inference-path network is based on a linear time algorithm for the satisfiability testing of prepositional Horn formula.

It is already indicated that the set of minimal supports for a query can be computed easily from the set of prime implicates of the RMS database. The negation of this minimal support clause becomes a solution hypotheses set for a given goal. Hence, by
such knowledge compilation, abductive synthesis of hypotheses is achieved very efficiently. Although the compilation of knowledge-base allows efficient abductive inference, the compilation process itself is very expensive in terms of computation time and memory.

Hence the core part of CMS as well as hypothetical reasoning is the computation of prime implicates. There have been many algorithms in the literature to compute the prime implicates of a set of clauses. These are the consensus method by Bartee et al [Bartee 62], methods by Karnaugh [Karnaugh 53], Quine [Quine 59], McCluskey [McCluskey 56], Kohavi [Kohavi 78], Semantic Resolution technique by Slagle et al [Slagle 70], Tison's method [Tison 67], Socher's method [Socher 91], [Jackson 92] etc. The next section gives a review of these methods.

Knowledge changes repeatedly from time to time and hence has to be updated and revised. Changes occur due to the addition, deletion, or change in the information. Due to the dynamic nature of the CMS, the most complicated, computationally expensive and essential operation is to update (RMS update problem) the existing database of prime implicates each time knowledge changes. Kean and Tsiknis' [Kean 90] proposed an incremental prime implicant algorithm (IPIA) that updates the set of prime implicates when the original corresponding knowledge is modified by addition of new knowledge which is a single clause. But, generally, knowledge may be a set of clauses rather than a single clause. In this case, IPIA is not efficient since each of the clauses in the set has to be treated one by one. The algorithm proposed in this thesis (Section 3.3.4) treats the set of clauses collectively, and computes the set of prime implicants for the updated knowledge. It can be seen that the proposed method is efficient both in global and incremental modes.
1.3 Different Methods for Compilation

The different methods to compute the prime implicants/implicates of a formula are briefly reviewed here.

Quine's method

There exist numerous methods for reducing any formula to its simplest equivalent. Quine [Quine 52, Quine 55, Quine 59] proposed a mechanical procedure based on the developed Disjunctive Normal Form. Prime implicants of a formula are used to express the formula in its simplest equivalent. It is shown in [Quine 52] that any simplest equivalent of a formula is a disjunction of prime implicants of the formula. Any given formula has to be converted into an equivalent developed formula in which all the clauses have all the variables either in the positive or negative form from the set of variables. The prime implicants of the developed formula are computed following a mechanical routine. Initially consider the list of clauses in the formula as the list of prime implicants. This list is extended according to the following principle: whenever two entries can be found in the list which are identical except for the negation sign, add their common part (consensus of the two entries) as a new entry in the list; check marks are applied to those two entries which generate the new entry. The check mark is not treated as a disqualification for further consideration of those clauses. The list is extended by this process as far as possible. The entries which do not bear any check marks give the list of prime implicants.

The shortest normal equivalent is obtained by deleting from the alternation of all its prime implicants the largest possible combination of jointly superfluous clauses. Though this method gives a mechanical procedure to compute the prime implicants of a formula, the method is not systematic. One of the major drawbacks of this method is that it generates repeated clauses, and requires a large number of basic operations. This
routine, though not unmanageable, turns out to be far more laborious than the method of merely locating and cancelling redundancies. Moreover, the two methods are almost independent. The laborious method of finding simplest normal equivalents depends on a preliminary expansion into a developed normal formula, and this expansion is not affected by any previous cancelling of redundancies.

Tison’s method

Tison [Tison 67] contributed a systematic method for determining the prime implicants of a Boolean function. The prime implicants are determined by generalization of the consensus operation which is performed systematically. It is to be noted that the method does not need the formula to be in the developed form as in the case of Quine’s method. Consensus is extended from two to any number of clauses. A property of these generalized consensus relations is that the consensus of two implicants of a formula gives another implicant. This property helps to find prime implicants systematically. The method is simple because the application of consensus of order two is no longer iterative as in Quine’s method.

One key note of Tison’s method is that of a biform variable: a variable which occurs both positively and negatively in the formula. For each biform variable \( x \) and for every pair of clauses \( D_i, D_j \) in the formula \( \mathcal{F} \), add the consensus of \( D_i \) and \( D_j \) with respect to the variable \( x \), denoted as \( \text{Con} (D_i, D_j, x) \), to \( \mathcal{F} \), (if such a consensus is possible) and delete every subsumed clause\(^1\) from the formula. When all possible consensus with respect to all possible biform variables are computed, and if all subsumed clauses are deleted, then \( \mathcal{F} \) will contain all the prime implicants of the original set of clauses, and only the prime implicants. The term consensus here is a restricted kind of resolution in which the attention is only to those resolvents of clauses in \( \mathcal{F} \) which are tautologies.

\(^1D_i \) subsumes \( D_j \) if \( D_i \subseteq D_j \)
The algorithm exploits the fact that each of the biform variable will be used exactly once in the algorithm. The consensus operation is equivalent to a resolution step and fundamentality test.

**Ordered clause consensus method**

The number of implicates grows exponentially in general as the resolution process proceeds, which is the serious problem in computing prime implicates. All the original and generated clauses remain to the end and become prime implicates if no subsumed clause appear during the resolution process. So the computational cost does not depend much on the ordering of the biform variables selected. However, the computational cost is influenced to a large extent by the order of the biform variable sequence used in resolution steps if there exists subsumed clauses which are to be deleted. Tsuruta and Ishizuka [Tsuruta 92] have developed a fast and efficient method named *Ordered Clause Consensus* (OCC) method for generating prime implicates utilizing the role of the biform variable. If a variable is a monofonn one in a set, there is no consensus so that no resolution takes place with respect to the variable, and hence monoform variables are not considered for resolution. All possible unit resolutions regarding every unit variable is performed and all subsumed original clauses are deleted. This does not increase the number of clauses since the resolvent clause always subsumes its one parent clause, which is deleted from the set of prime implicates. This method uses a heuristic regarding the resolution order of biform variables while generating the prime implicates through successive steps. The number of consensi possible with each of the biform variable is arranged in increasing order. The heuristic used is; *the biform variable corresponding to the least number* is considered; and consensi with respect to this variable are performed. This helps the set of prime implicates to grow slowly at the early stages, which in turn reduces the computational cost as reported in [Tsuruta 92].


**Incremental method**

All these methods that generate prime implicants/implicates are applicable to the RMS update problem. However, they are inefficient simply because they are concerned with the generation of prime implicants/implicates from an arbitrary Boolean expression. What is needed is an incremental method which generates the prime implicants/implicates using the already available prime implicants/implicates of the original formula and the new formula when the original Boolean expression is modified.

More formally, if the prime implicates denoted by $\Pi(F)$ of a formula $F$ in the conjunctive form are known, the task is to find the set of prime implicates of $FA H$ where $H$ is another formula in the normal conjunctive form. Obviously, the prime implicates of $FA H$ can be generated directly from $FA H$ using any of the known conventional methods discussed above. But this results in not utilizing the $\Pi(F)$ which is already available. Hence, ideally, one would like to generate the set of prime implicates of $FA H$ from the $\Pi(F)A H$. Using the conventional methods to compute prime implicates of $\Pi(F)A H$ results in lot of redundant computations simply because all the conventional methods do not exploit the fact that the elements of $\Pi(F)$ are already prime. Hence, an algorithm which can compute the prime implicates incrementally, and which reduces the redundant computations, is appreciable.

The IPIA proposed by Kean and Tsiknis [Kean 90] computes the prime implicates of $FA H$ from $\Pi(F)$ and $H$ when $H$ is a single clause $h$. This method resembles Tison’s method [Tison 67] except that it stores the new implicates of $\Pi(F)U \{h\}$ in a new set. Incremental computation admits two simplifications which are (i) it needs only to perform consensus with respect to biform variables occurring in the input clause $i$; and (ii) it needs only to perform consensus between clauses from the sets $H$ and $\Pi(F)$, but not within the same set since the clauses in $\Pi(F)$ are already prime.
A consensus tree is constructed with the input clause \( h \) as the root, and every arc is labelled by a clause from \( \Pi(\mathcal{F}) \), and every node (except the root) is labelled by the consensus of its parent and the associated arc label. Label of nodes are the implicates, and subsumption among these labels are performed to obtain the prime implicates. Various optimization techniques have also been proposed to improve IPIA. However, this method suffers from few drawbacks. In order to overcome the drawbacks of IPIA, Jackson [Jackson 92] proposed another algorithm which also resembles Tison's method, except that it computes prime implicates using a particular resolution strategy which concentrates on finding merges. Merges are resolutions involving a pair of clauses that contain literals of the same sign in addition to complimentary literals. A cost-function defined biases the search towards consensi that generate merges. The resulting merge contains fewer literals than non-merge resolvents derived from parents of the same size. This method computes the compliments of clauses of \( \mathcal{F} \) in a particular order so as to avoid the duplication of steps.

**Slagle's method**

All the methods discussed so far compute the prime implicants when the formula is in disjunctive form, and the prime implicates when the formula is in the conjunctive form. Different from all these methods, Slagle [Slagle 79] describes a method which determine the prime implicants of a formula which is in the conjunctive form. The algorithm works as follows: All the clauses in the formula containing a complementary pair of literals are deleted. A *semantic tree* is constructed keeping this set of clauses thus obtained at the initial node. The literal which occurs in more number of clauses of the formula is given a higher frequency. *Sprouting* from the initial node with the frequency ordering is performed. If there is a nonterminating node, choose an ordering for that node and repeat the process. This sprouting is repeatedly done until there is
no nonterminating nodes to do sprouting. For each success node in the semantic tree, collect the product of all the literals at the branches on the path from the top down to the success node of the semantic tree. The set of all such products thus obtained gives all the prime implicants of the formula. The algorithm may possibly generate some non prime implicants. However, the use of frequency ordering of literals, helps to generate very few (possibly none) non prime implicants. The algorithm may also be used to find the minimal sums of a Boolean function. Reiter and de Kleer [Reiter 87] suggested this method as a well-disciplined method to compute the prime implicants. However, there are better methods to compute the prime implicants.

Though Tsuruta and Ishizuka [Tsuruta 92] also discussed frequency ordering, it has to be noted that these two orderings are different. Slagle considers all the literals for frequency ordering while Tsuruta and Ishizuka consider all the variables and the number of consensi possible with respect to this variable to fix frequency ordering. In Slagle's algorithm, the frequency ordering of literals at each node is required in order to achieve efficiency. But this is not the case with the ordering which Tsuruta and Ishizuka applies. The same ordering can be applied to the IPIA of Kean and Tsikins'. But, when the original formula is appended with more than one clause, computation of the prime implicants is not possible by performing the algorithm just once. The algorithm has to be performed as many times as there are clauses in the new input formula. This approach is obviously not efficient and hence a method which can handle the case when the original formula is appended with another formula will be of great use.

Socher [Socher 91] proposed an algorithm similar to that of Slagle et al [Slagle 70]. A concept of path in a matrix is introduced and shown that the set of prime paths is a matrix gives the set of prime implicants of a formula. The main difference is in the data structure used. Slagle uses semantic tree as the basic data structure while the data structure used in Socher's method is matrix. This makes Socher's method very suitable for
application in matrix methods for automated theorem proving. Though this method has advantages over other methods, it is not suitable for incremental computation of prime implicants. Apart from this, the algorithm performs certain redundant computations. This is discussed in detail in Chapter 2.

Present work

It is already seen that the prime implicants/implicates are significant in the context of RMS as well as hypothetical reasoning. Though there are different techniques to compute the prime implicants/implicates, as already indicated above, all techniques suffer from one drawback or the other. Hence there is a need to develop an efficient algorithm which overcomes the drawbacks of earlier methods. Besides applications in reasoning, prime implicants/implicates find applications in several areas such as switching theory, combinatorial optimization, computer vision etc. It would be difficult to design a very efficient algorithm for this hard problem suitable to all the applications. This prompts us to design a new algorithm which will overcome the drawbacks of earlier methods in the context of RMS. This thesis is concerned with the design of RMS based on the framework proposed by Reiter [Reiter 87] in CMS and also with the design of an algorithm to compute the prime implicants/implicates in the global as well as in the incremental mode. In this process, a new knowledge representation scheme is proposed. Using the divide-and-conquer paradigm, an efficient technique for knowledge compilation for the purpose of RMS is proposed in this thesis. The efficiency of the method hinges on the tree-structure representation for prepositional clauses, which also helps in a novel RMS design. The formula is subdivided into two subformulas and the set of prime implicants for both the subformulas are computed from where the prime implicants for the formula are computed. The prime implicants are maintained in a binary tree.

Unlike the earlier algorithms which are inherently sequential, the algorithm proposed
in this dissertation is naturally parallelizable. A parallel knowledge compilation technique, designed because of the special characteristic of PIAP, is an efficient tool for parallel RMSs.

1.4 Organization of the Thesis

The thesis comprising seven chapters is organized in the following manner:

In Chapter 2, the binary matrix representation of a formula and the three algorithms [Tison 67, Kean 90, Socher 91] to compute the prime implicants/implicates are explained using examples. The paths in a matrix are characterized and based on these characterizations, a new Prime Implicant Algorithm using Paths (PIAP) is proposed in Chapter 3. Theoretical arguments supporting the algorithm are also given in the same chapter.

The tree-structure representation of a prepositional formula suitable for the algorithm PIAP is presented in Chapter 4. The implementation details of PIAP and the experimental results that substantiate the theoretical arguments in Chapter 3 are presented in Chapter 4. Further, different methods to update a knowledge base, and to compile the knowledge incrementally, are discussed. Chapter 5 deals with the parallel algorithm to compute the prime paths of a formula.

In Chapter 6, an application of the results to computer vision is discussed. It is shown that reconstructing the three-dimensional shape from multiple silhouettes can be formulated as a problem in prepositional logic. All possible reconstructions of the object can be obtained since the problem of shape from silhouettes is viewed as the problem of computing the prime paths of a formula. It is also shown that this method provides a better way of shape reconstruction. Chapter 7 summarises the contributions and limitations of the work reported in this dissertation and considers possible routes for further research work.