A Logically Complete Reasoning Maintenance System Based on a Logical Constraint Solver

J.C. Madre and O. Coudert

Bull Corporate Research Center

Rue Jean Jaures

78340 Les Clayes-sous-bois   FRANCE

Abstract

This paper presents a logically complete assumption based truth maintenance system (ATMS) that is part of a complex blast furnace computer aided piloting system [5]. This system is built on an efficient and logically complete propositional constraint solver that has been successfully used for industrial applications in computer aided design.

1 Introduction

A reasoning maintenance system (RMS) is a critical part of a reasoning system, since it is responsible for assuring that the inferences made by that system are valid. The reasoning system provides the RMS with information about each inference it performs, and in return the RMS provides the reasoning system with information about the whole set of inferences.

Several implementations of reasoning maintenance systems have been proposed in the past, remarkable ones being Doyle's truth maintenance system (TMS) [6], and De Kleer's assumption-based truth maintenance system (ATMS) [7]. Both of them suffer from some limitations. The TMS considers only one state at a time so it is not possible to manipulate environments [6]. The ATMS is intended to maintain multiple environments, but it is limited to propositional Horn clauses [7], so it is not logically complete with respect to full standard Propositional Logic.

This paper describes an original assumption-based truth maintenance system that is logically complete. The power of this system comes from the remarkable properties of the *typed decision graph canonical* representation of propositional formulas [I]. This representation originates from researches made in the field of formal verification of hardware. It has been successfully used in PRIAM [10], a formal verifier of digital circuits that has been integrated in Bull's CAD system in 1988, and is now used by all VLSI circuit designers at Bull.

The paper is divided in 4 parts. Part 2 gives the logical specifications of the reasoning maintenance system. Part 3 presents the typed decision graph representation. Part 4 describes the implementation of the RMS based on this representation. Part 5 discusses the experimental results that have been obtained with this system.

2 Functional Specifications

The reasoning maintenance system receives *prepositional formulas* from the reasoning system [12]. These prepositional formulas are built out of a countably infinite set of propositional variables using the usual logical connectors $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ and $\neg$. We note by *KB* the *knowledge base* of the system, which is the set of propositional formulas that the RMS maintains. The RMS distinguishes two kinds of variables, *assumptions* and *data,* the assumptions being the variables from which the data will be deduced. An *environment* is a set of *literals,* each literal being either a propositional variable or its negation.

Since any formula is a logical consequence of an inconsistent set of formulas, the first task of the RMS is to keep the knowledge base *KB consistent* If $KB = \{f_1,...,\ f_n\}$ then it is consistent if and only if there exists at least one assignment *of truth* values *to* the variables of $f_1,..., f_n$ for which these formulas all evaluate to *True.* A consistent environment *E* is consistent with *KB* if *(KB $\cup$ E)* is consistent. If this is not the case, the RMS can be asked to compute the *maximal consistent sub-environments* of *E* that are all the maximal subsets of the set *E* consistent with *KB.*

It is critical for the reasoning system to know, at any moment, which data can be believed and which cannot be. The knowledge base *KB* and the environment *E* are said to *support* the datum *d* if and only if *d* is a logical consequence of the formulas in *(KB $\cup$ E),* which is expressed by: $KB,\ E \models d.$ If this is the case, the RMS can be asked to compute all the minimal subsets of the environment *E,* called the *minimal supporters,* that also support this datum. Finally the *labels* of a datum *d* are all the minimal environments made of assumption variables that support *d.*

3 Typed Decision Graphs

Several normal forms of Propositional Logic have been used in the past as the basis of automated propositional provers and constraint solvers [II]. Some of these normal forms have the remarkable property of being *canonical.* These canonical forms are very appealing because they support very simple proof procedures. Two formulas written in canonical form are equivalent if and only if they are syntactically equal. More generally, since Propositional Logic is sound and complete, the formula

g can be deduced from the assumptions $\{h_1, \ldots, h_n\}$, which is expressed $\{h_1, \ldots, h_n \models g\}$ if and only if the formula $((h_1 \wedge \cdots \wedge h_r \Rightarrow g)$ is valid, which is the case if and only if its canonical form is equal to *True*.

The computational cost of a proof procedure based on canonical rewriting depends on the time needed to rewrite the formula to be proved valid into its canonical form. This time is in the worst case. for all canonical forms, exponential with respect to the size of the formula to be rewritten. The problem is that this worst case was very easily obtained for all the canonical forms that were known until very recently [10]. We present here a new canonical representation that has been shown by experience to support a very efficient and simple rewriting process.

Typed decision graphs (TDG) are a new *canonical graph representation* of Propositional Logic [I]. This canonical form is directly inspired by Shannon's decomposition theorem. This theorem states that any Boolean function $/(a_i, \ldots, r_n)$ from the set $\{0,1\}^n$ to the set $\{0,1\}$ can be expressed in terms of a unique couple of Boolean functions $(/o,/i)$ from the set $\{0,1\}^{n \wedge}$ to the set $\{0,1\}$ in the following way:

$$ / = (\neg x_1 \wedge f_0) \vee (x_1 \wedge f_1). $$

The decomposition or expansion process defined above can be iterated until all variables are eliminated. It produces, for any Boolean function $/$ and for any propositional formula $/$, a unique decomposition tree, called Shannon's tree of$/$, whose leaves are the constant *False and True*. The problem is that the decomposition tree of a formula f with n variables has 2^{n-1} vertices and 2^n leaves so it rapidly becomes too large to be computed when *n* becomes large. However in many cases, a lot of these vertices are redundant and can thus be eliminated. The elimination rule is very simple: a vertex is useless if and only if its left and right subtrees are identical, and it can then be replaced by either of these subtrees. R. E. Bryant showed in [3] that by sharing identical subtrees that are embedded at different places in Shannon's tree of a formula, the memory space required to store this tree can be dramatically reduced, and that the resulting graph representation, called the *binary decision diagram* (BDD) is canonical.

Shannon's tree of a formula f and its binary decision graph are canonical with respect to the decomposition ordering that has been chosen. The size of this graph, defined as the number of its vertices, heavily depends on this variable ordering [3], For instance there exist formulas that have a BDD whose size is linear with respect to the size of the formula for the best variable ordering and exponential for the worst variable ordering. Though finding the best variable ordering is a NP-complete problem [9], good heuristics have been proposed to compute a good variable ordering from the structure of the syntactic tree of the formulas [13].

More recently, J. P. Billon defined a new canonical graph representation of Propositional Logic, called the *typed decision graphs* (TDG), that holds all the remarkable properties of the binary decision diagrams in addition with the *instantaneous negation* [I]. Instantaneous

negation is obtained by using the same graph to represent a formula and its negation, and by using typed edges in the graphs: a positive edge is a standard one, a negative edge indicates that the pointed graph has to be transformed by a negation operation to obtain the equivalent BDD. Figure 1 shows the BDD and the TDG of the formula $f = (a \wedge (c \oplus d)) \vee (b \wedge (c \Leftrightarrow d))$. Typed decision graphs as well as binary decision diagrams support very efficient proof procedures, as well as efficient resolution procedures of Boolean equations [11].
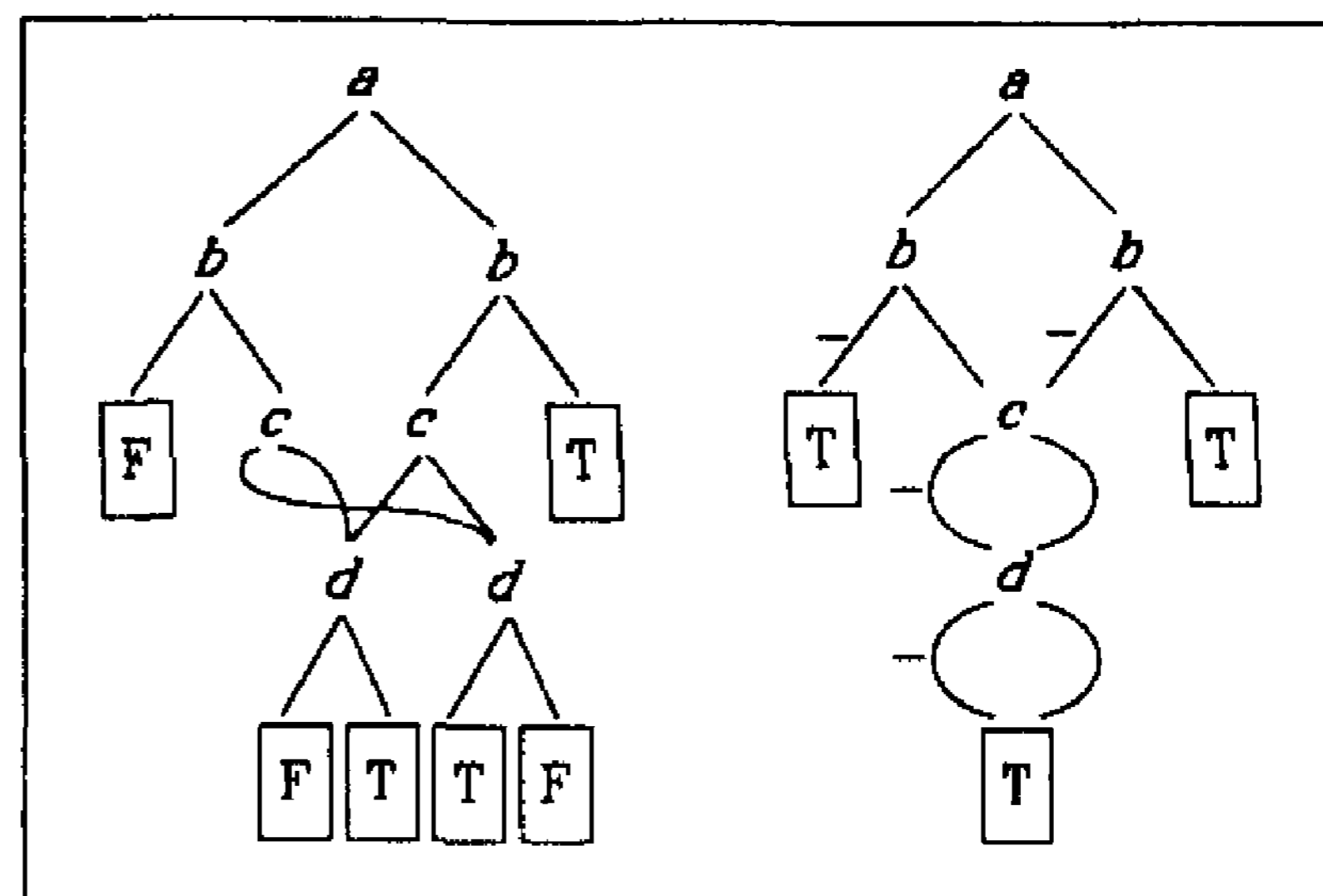

Figure 1: BDD and TDG of the formula
$$ f = (a \wedge (c \oplus d)) \vee (b \wedge (c \Leftrightarrow d)) $$

Typed decision graphs have been developed in the framework of a research project on formal verification of digital circuits. The automated propositional prover that has been built on the typed decision graphs is the kernel of an industrially used formal verifier of circuits called PRIAM [10]. This tool, which has been integrated in Bull's proprietary computer aided design system in 1988, automatically proves the correctness of circuits with respect to their specification,

4 Implementation of the RMS

This section explains how the typed decision graph representation makes easy and efficient the implementation of the reasoning maintenance system. It gives, for each of the basic tasks that the system must perform, the corresponding procedure and its computational complexity.

4.1 Computing the TDG of a formula

The first task of the system is to compute the typed decision graphs of the formulas that it receives from the reasoning system. The rewriting process takes as input a propositional formula. It performs a depth first traversal of the syntactic tree of this formula, and it computes its TDG in a bottom-up manner by combining the typed decision graphs of its embedded sub-formuias. The leaves of the syntactic tree of the formula are variables whose associated typed decision graphs have only one vertex. These elementary graphs are then combined using the algorithm given in [3], that has been extended to TDG's, until the root of the syntactic tree is reached. The complexity of this rewriting process is in the worst case exponential with respect to the size of the syntactic tree of the formula to be rewritten.

4.2 Keeping the Knowledge Base Consistent

Keeping the knowledge base consistent is trivial. Each time a formula f has to be added to the knowledge base KB, the system first determines whether this formula is consistent, which is the case if and only if its TDG is different from the constant *False*. The system then determines whether this formula can be added to KB without making it becoming inconsistent. If $KB = \{f_1, \ldots, f_n\}$, the resulting base is consistent if and only if the formula $(f_1 \wedge \cdots \wedge f_n \wedge f)$ is satisfiable, which is true if and only if its TDG is not equal to the constant *False*.

It would be very costly to recompute, at each addition of a formula to KB, the TDG that represents the conjunction of the n formulas that the system holds. Consequently, the system maintains the TDG G_n of this conjunction, which is initially equal to *True*, and updates it at each addition of a formula. The cost of adding the formula f into the knowledge base KB is so made of two parts. First the system must compute the TDG of f. It must then compute the typed decision graph of G_{n+1}, which cost is in $O(|G_n| \times |G_f|)$, where $|G_n|$ and $|G_f|$ are the numbers of vertices in the TDGs of G_n and of G_f respectively.

4.3 Making Proofs

Thanks to the canonicity of the graph representation, proofs are also very easy to perform. Consider a literal d, the knowledge base $KB = \{f_1, \ldots, f_n\}$ and the environment $E = \{h_1, \ldots, h_k\}$. The environment E is consistent if and only if the typed decision graph of the formula $(h_1 \wedge \cdots \wedge h_k)$ is different from *False*. In the same way, the environment E is consistent with the knowledge base KB if and only if the formula $C = (f_1 \wedge \cdots \wedge f_n \wedge h_1 \wedge \cdots \wedge h_k)$ does not reduce to *False*. Finally the literal d holds in the environment E of KB if and only if the formula $D = (C \Rightarrow d)$ is valid, which happens if and only if its TDG is equal to *True*.

Computing the TDG of the formula C is in $O(|G_n| \times k)$ where G_n is the TDG of the conjunction of all the formulas in KB. Since the TDG of the literal d has only one vertex, computing the TDG of the formula D is linear with respect to the number of vertices in the TDG of C.

4.4 Minimal Suppporters

Consider the knowledge base $KB = \{f_1, \ldots, f_n\}$ and the environment $E = \{h_1, \ldots, h_k\}$ that support the literal d. The minimal supporters problem is to find all the minimal subsets E' of E that also support d.

Without loss of generality, we will here assume that the literals $h_1, \ldots, h_k$ and d are all positive, i. e. they are propositional variables. If this is not the case we create, for each negative literal l occurring in $\{h_1, \ldots, h_k\} \cup \{d\}$, a new variable pl, then we add the formula $(pl \Leftrightarrow (\neg l))$ into the knowledge base, and finally we substitute l with pl in E. The renaming technique defined above allows us to assume, too, that the literal d is not in the set $\{h_1, \ldots, h_k\}$.

A minimal supporter E' of d is a set $\{h'_1, \ldots, h'_m\}$, where $h'_1, \ldots, h'_m$ are elements from the set $\{h_1, \ldots, h_n\}$, such that: $KB, E' \models d$. We note by P the

conjunction $(f_1 \wedge \cdots \wedge f_n)$, and by H' the conjunction $(h'_1 \wedge \cdots \wedge h'_m)$. The formula given above can be rewritten into $(P, H' \models d)$, which is true if and only if the formula $((P \wedge H') \Rightarrow d)$ is valid.

Shannon's expansion of the formula P with respect to the variable d provides us with a couple of formulas (P_0, P_1) such that $(P \Leftrightarrow ((\neg d \wedge P_0) \vee (d \wedge P_1)))$. Since the variable d does not occur in H', Shannon's expansion of the formula $(P \wedge H')$ with respect to the variable d is the couple of formulas $((P_0 \wedge H'), (P_1 \wedge H'))$. Finally, Shannon's expansion of the formula $((P \wedge H') \Rightarrow d)$ with respect to d is $(\neg (P_0 \wedge H'), True)$ and this formula is a tautology if and only if the formula $(\neg (P_0 \wedge H'))$ is valid. This means that the minimal supporters problem comes down to finding all the minimal products H' such that the formula $(\neg (P_0 \wedge H'))$ is valid.

These minimal products can be computed in two steps. The first step consists in computing the characteristic function χ_{min} of the set of environments built out of the variables $h_1, \ldots, h_k$ that support the literal d. The second step consists in choosing amongst all these sets the minimal ones.

The variables occurring in the formula P_0 are all the variables that have been introduced in the reasoning maintenance system except for the variable d. These variables are the variables $h_1, \ldots, h_k$ and some other variables $x_1, \ldots, x_p$. The first step in the computation of the minimal supporters of d is to get rid of the variables $x_1, \ldots, x_p$ that are not useful for this computation. Any assignment of Boolean values $v_1, \ldots, v_k$ to the variables $h_1, \ldots, h_k$ uniquely defines an environment built out of these variables. This environment supports the datum d if and only if the formula $(\forall x_1 \ldots x_p \ P_0[h_1 \leftarrow v_1] \ldots [h_k \leftarrow v_k])$ is valid. The quantified part of this formula expresses that the formula $(P_0[h_1 \leftarrow v_1] \ldots [h_k \leftarrow v_k])$ evaluates to 1, whatever values are assigned to the variables $x_1, \ldots, x_p$. Using this notation, we can define the characteristic function χ_{min} with the equation:

$$\chi_{min}(h_1, \ldots, h_k) = (\forall x_1 \ldots x_p \ P_0)$$

The TDG G_{min} of χ_{min} is directly obtained from the TDG of P_0 by eliminating the variables $x_1, \ldots, x_p$. This elimination is based on the identity:

$$(\forall x_1 \ldots x_p \ f) = (\forall x_1 \ldots x_{p-1} \ (f_0 \wedge f_1))$$

where the couple of formulas (f_0, f_1) is Shannon's expansion of f with respect to the variable x_p.

The second step in the minimal supporters computation consists in computing the minimal products H' such that $(\neg (P_0 \wedge H'))$. These minimal products are by definition the prime implicants of the function χ_{min} [11], [14]. Finding the prime implicants of a Boolean function f written in normal disjunctive or in normal conjunctive form is a difficult problem [2]. This problem is made much simpler when the function is written in Shannon's canonical form thanks to Shannon's decomposition theorem. We propose in [11] the function Primes for computing all the prime implicants of the function f that is based on a depth first traversal of the TDG of the function f and treats each vertex of this graph only once.

This algorithm is a modified version of the algorithm given in [2] for functions in disjunctive normal form.

Note that there exist Boolean functions whose TDG has n vertices and that have 2^n prime implicants. This shows why typed decision graphs are a good representation of prepositional formulas, since it means that there are formulas that have an exponential size with respect to the number of their variables when represented in disjunctive or conjunctive (clausal) normal form and that are represented by a TDG of polynomial size.

The computational cost of the minimal supporters computation is the following. Computing Shannon's decomposition of the formula P with respect to the variable d is linear with respect to the size of the TDG of the formula P. Computing the characteristic function x min is in the worst case exponential with respect to the size of the TDG of the formula P_0, but this worst case has never occurred in real world problems. Finally applying the function Primes can also be in the worst case exponential with respect to the size of the TDG of the formula Xmin- However in this case this exponential complexity does not depend on the typed decision graph representation but on the size of the set to be computed.

The set of prime implicants that are generated by the function **Primes** for the different vertices of the graph drawn in Figure 2 are the following: $\{(\neg d)\}$ for A, $\{(d)\}$ for B and C, $\{(\neg d)\}$ for D and E, $\{(\neg c),(\neg d)\}$ for F, $\{((\neg c) \wedge d)\}$ for G, $\{((\neg c) \wedge d), (c \wedge (\neg d))\}$ for H, $\{(c \wedge (\neg d))\}$ for I, $\{((\neg c) \wedge d), ((\neg b) \wedge (\neg c)), ((\neg b) \wedge (\neg d))\}$ for J, $\{(c \wedge (\neg d)), ((\neg b) \wedge (\neg c) \wedge d)\}$ for K and $\{((\neg b) \wedge (\neg c) \wedge d), ((\neg a) \wedge (\neg b) \wedge (\neg c)), ((\neg a) \wedge (\neg b) \wedge (\neg d)), ((\neg a) \wedge (\neg c) \wedge d), (a \wedge c \wedge (\neg d))\}$ for L.
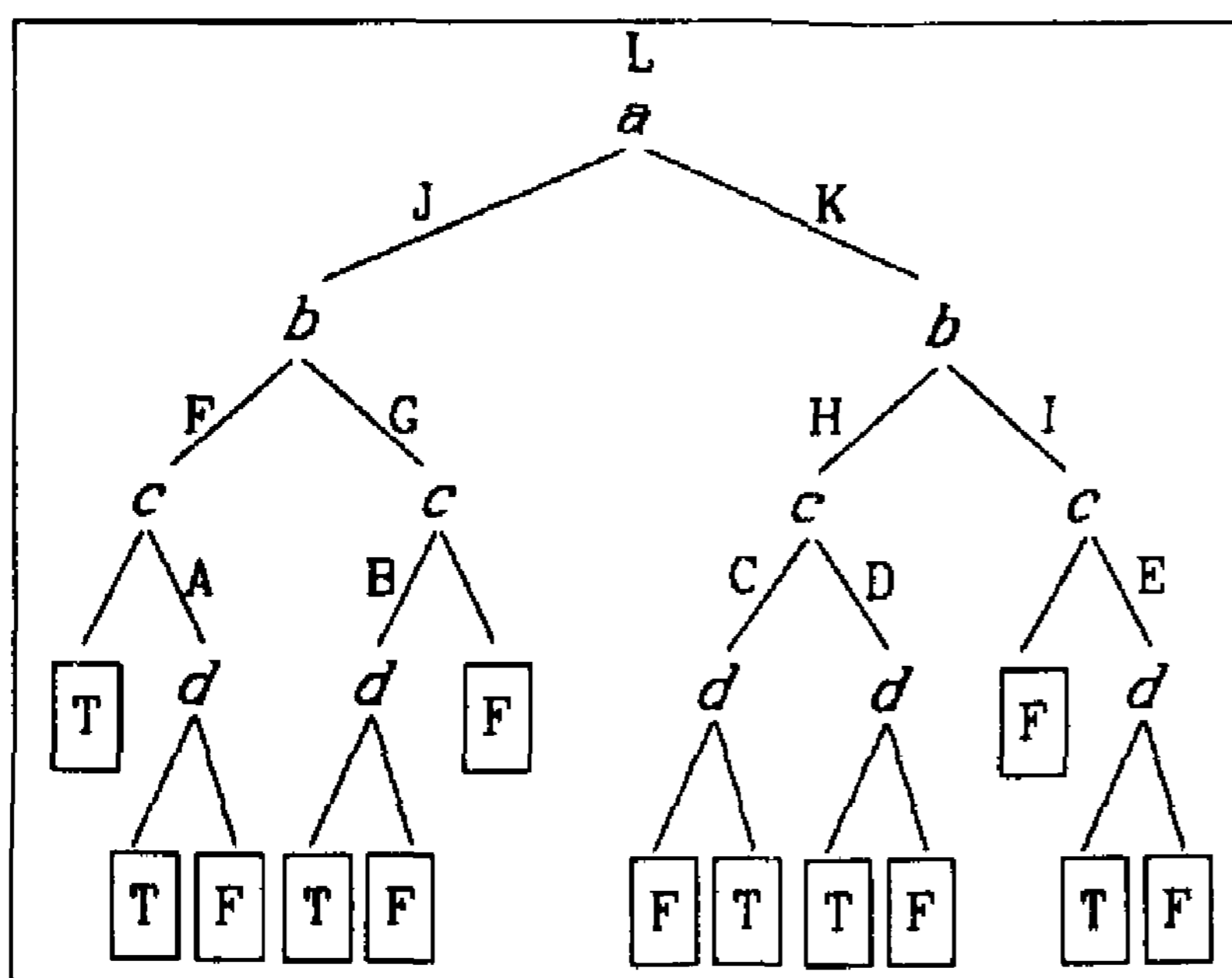

Figure 2: A sample execution of the function **Primes**

4.5 Maximal Consistent Sub-Environments

Consider the knowledge base $KB = \{f_1, \ldots, f_n\}$ and the consistent environment $E = \{h_1, \ldots, h_k\}$ that is inconsistent with KB. The maximal consistent sub-environments problem is to find all the maximal subsets E' of E that are consistent with KB. Without loss of generality, we will assume, like we did in Section 4.4, that the literals $h_1, \ldots, h_k$ are positive, i. e. they are

propositional variables.

By definition, a maximal consistent sub-environment of E is a subset $E' = \{h'_1, \ldots, h'_m\}$ of E such that E' is consistent with KB et there does not exist a subset E'' of E such that $(E' \subset E'')$ and E'' is consistent with KB. If we note by P the conjunction $(f_1 \wedge \cdots \wedge f_n)$, and by H' the conjunction $(h'_1 \wedge \cdots \wedge h'_m)$, then E' is a maximal consistent sub-environment of E if and only if the formula $(P \wedge H')$ is satisfiable, which is the case if and only if its TDG is not the constant False. The problem thus comes down to finding all the maximal sub-products H' of the product $(h_1 \wedge \cdots \wedge h_m)$ such that $(P \wedge H')$ is satisfiable.

These maximal sub-products can be computed using an algorithm that is very similar to the algorithm given in Section 4.4 for the minimal supporters problem. This algorithm works in two steps. The first step consists in computing the characteristic function χ_{max} of the set of environments build out of the variables $h_1, \ldots, h_k$ that are consistent with the knowledge base KB. The variables occurring in the formula P are all the variables that have been introduced in the reasoning maintenance system. These variables are the variables $h_1, \ldots, h_k$ and some other variables $x_1, \ldots, x_p$. Using the notation defined in Section 4.4, we can define the function χ_{max} with the equation:

$$\chi_{max}(h_1, \ldots, h_k) = (\exists x_1 \ldots x_p \, P)$$

which expresses that $\chi_{max}(v_1, \ldots, v_k)$ is equal to True if and only if there exists at least one assignment of Boolean values to the variables $x_1, \ldots, x_p$ for which the formula $(P[h_1 \leftarrow v_1] \ldots [h_k \leftarrow v_k])$ evaluates to 1.

The second step in the maximal sub-environments computation consists in computing the maximal products H' such that $(\chi_{max} \wedge H')$ is satisfiable. If the function χ_{max} is the constant False, then there are no such products, which means that the set of maximal consistent sub-environments is empty. If the function χ_{max} was equal to True, then E itself would be the only one maximal consistent sub-environment. Since we assumed that E is inconsistent with KB this situation can not happen. The maximal products searched for can be computed with the function **MaximalProducts** given in [11] using a depth first traversal of the TDG G_{max} of the function χ_{max} [11]. During this traversal, the function **MaximalProducts** traverses each vertex of the graph G_{max} only once, and it computes the maximal products associated with this vertex by combining the sets of maximal products that have been obtained for its left and right branches [11]. The cost of the maximal consistent sub-environments computation is the same as the computational cost of the minimal supporters computation.

The set of maximal cubes generated by the function **MaximalProducts** for the different vertices of the graph drawn in Figure 3 are the following: $\{True\}$ for A and B, $\{(d)\}$ for C, $\{True\}$ for D, $\{(c)\}$ for E, $\{(c \wedge d)\}$ for F, $\{(d)\}$ for G, $\{(b \wedge c)\}$ for H, $\{(b \wedge d), (c \wedge d)\}$ for I, $\{(b \wedge c) \text{ and } (a \wedge b \wedge d), (a \wedge c \wedge d)\}$ for J.
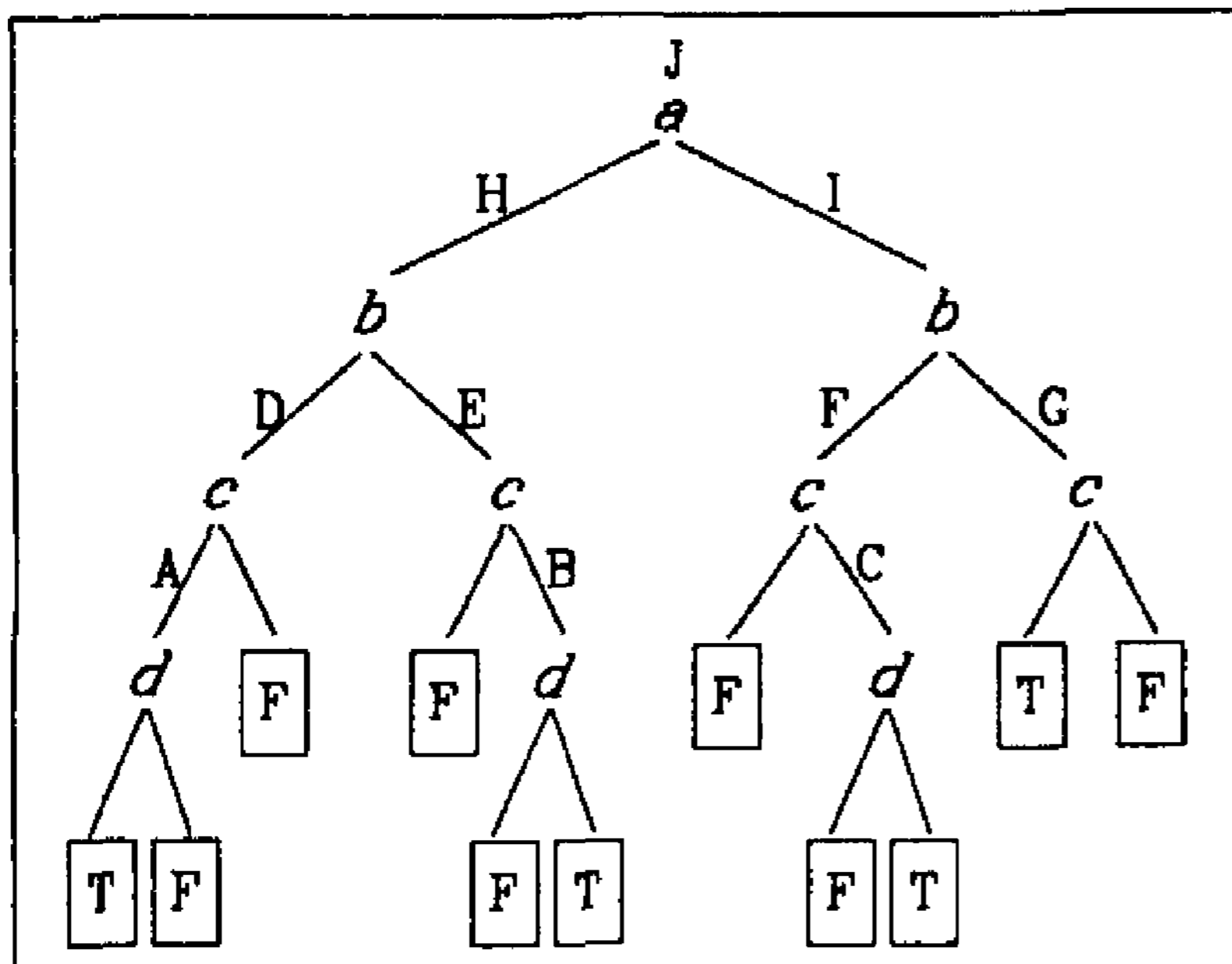
Figure 3: A sample execution of the function
MaximalProducts

4.6 Labels

The variables that compose the formulas of the knowledge base $KB = \{f_1, \ldots, f_n\}$ are the assumption variables $a_1, \ldots, a_p$ and the datum variables $d_1, \ldots, d_q$. By definition the datum variables have been deduced from the assumption variables. The labels problem is to find all the minimal environments made of assumption variables that support one of the datum variables.

The labels of the datum d are computed in three steps. The first step consists in computing the characteristic function X_{max} of the set of environments built out of the assumption variables that are consistent with the knowledge base KB. This characteristic function, defined by:

$$\chi_{max}(a_1, \ldots, a_p) = (\exists d_1 \ldots d_q \ P)$$

where we note by P the conjunction $(f_1 \wedge \cdot \wedge f_n)$, can be computed using the procedure defined in Section 4.5. The second step consists in computing the characteristic function X_{min} of all the environments built out of the assumption variables that support the datum d. This characteristic function can be computed using the procedure given in Section 4.4. Finally the third step in the labels computation consists in extracting these labels from X_{max} and X_{min}. There are two ways to perform this extraction, depending on whether the characteristic function X_{max} is used *passively* or *actively*:

- The minimal environments that support the datum d can be extracted from the TDG of X_{min} using the function Primes presented in Section 4.4. Some of these minimal environments can be inconsistent with the knowledge base KB. These minimal environments that are inconsistent with KB can be eliminated using the characteristic function X_{max} as *a. filter*. In this case the characteristic function X_{max} is used passively because all minimal environments that support d are computed, and inconsistent ones are eliminated afterwards.

♦ Inconsistent environments can be eliminated as soon as possible during the computation of the prime im-

plicants of the characteristic function X_{min}- The function Primes can be modified to take a second argument that is the TDG of the characteristic function X_{max}- During the bottom up generation process, the TDG of X_{max} is used to eliminate the inconsistent environments as soon as they are discovered.

The complexity of this procedure is the same as the complexity of the procedure Primes given in Section 4.4.

5 Experimental Results

The reasoning maintenance system presented in this paper has been written in the C language and the results given have been obtained on a Sun Sparc Station. The reasoning maintenance system has an integrated garbage collector that guarantees that the memory use is kept minimal during execution. This garbage collector is incremental which allows the RMS to be used in a real time environment.

The first set of examples given here are minimal supporters computations. We provided the RMS with n environments that support the datum p_1, and n environments that support the datum q. The environment $\{P, q\}$ supports the datum r. Each of these environments is made of n distinct assumptions variables. This means that the number of assumptions variables used in each of these examples is equal to $2 \times n$. The problem is to find the n^2 minimal supporters of r. Table 1 shows the results for these examples. It gives, for each example, the CPU time (CPU time) needed to compute the solutions, the number of vertices ($Size_{KB}$) of the TDG that denotes the conjunction of all the formulas in the knowledge base and the number of minimal supporters (Supporters) to be found. Note that the size of this graph and the CPU time needed to build the knowledge base and to compute the solutions grow linearly with n.

n	CPUTime	$Size_{KB}$	Supporters
5	1.2s	83	25
6	1.5s	116	36
7	1.8s	155	49
8	2.1s	200	64
9	2.6s	251	81
10	3.1s	308	100

Table 1: Applying the ATMS to the
minimal supporters problem

It has been shown that an ATMS can be used as a constraint solver [8]. An example of the use of an ATMS as a constraint solver is the n queen problem. This problem is to place n queens on a n x n chessboard so that no queen dominates any other one. In order to specify the n queen problem, we introduce n^2 propositional variables; each one is associated with one square of the chessboard and indicates whether there is a queen on this square. The formulas or constraints of the n queen problem express that there are no more than one queen on each line, each column and each diagonal of the chessboard.

Table 2 gives the results obtained for $n = 4$ to n = 9. For each of these problems, the table gives the CPU time (Times) needed to build the knowledge base, the size

(Size$_{KB}$) of the typed decision graph that represents this base, the CPU time (Time$_S$) required to compute the solutions and the number of solutions (Solutions), These solutions are obtained through a maximal consistent sub-environments request. Note that in this case the size of the TDG that represents the knowledge base grows exponentially with n, while the time needed to compute the solutions grows linearly with the number of solutions to be found.

Queens	Time$_B$	Size$_{KB}$	Time$_S$	Solutions
4	0.2s	29	0.01s	2
5	0.6s	166	0.1s	10
6	1.0s	129	0.05s	4
7	2.7s	1098	0.4s	40
8	7.4s	2450	1.0s	92
9	33.5s	9556	7.0s	352

Table 2: Applying the ATMS to the n-queen problem

6 Conclusion

This paper has presented an original implementation of a logically complete assumption based truth maintenance system. This implementation is based on a new canonical representation of Boolean functions called the typed decision graphs. This canonical form, which is amongst the most compact representations of Boolean functions that are currently known, has remarkable properties that give the system its efficiency. Our approach is quite different from the one followed by De Kleer who proposes an ATMS as a constraint solver [8], since we use here a constraint solver developed for other purposes to build an ATMS. This ATMS has been integrated in the prototype version of a complex blast furnace computer aided piloting system [5]. Within this system the RMS receives formulas from a reasoning system written in Kool [4]. The piloting system is intended to be a real time system so an incremental garbage collection scheme has been developed that guarantees a continuous operation of the RMS.

Acknowledgments

References

[1] J. P. Billon. Perfect Normal Forms For Discrete functions. Bull Research Report N°87019, June 1987.

[2] R, K. Brayton, G. D. Hatchel, C. T. McMullen and A. L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, 1984.

[3] R. E. Bryant. Graph-based Algorithms For Boolean Functions Manipulation. *IEEE transactions on computers,* C35(8):677-691, August 1986.

[4] Bull Cediag. *Kool V2 Reference Manual* Bull Cediag, June 1989.

[5] Bull, ITM1, CSI. *Consultation pour la realisation d'un systeme d'aide a la conduite des hauls foumeaux. Offre technique et Organisationnelle.* Projet Sachem, October 1990.

[6] J. Doyle. A Truth Maintenance System. *Artificial Intelligence,* 12:231-271, 1979.

[7] J. de Kleer. An Assumption-based TMS. *Artificial Intelligence,* 28:127-162, 1986.

[8] J- de Kleer, Comparison of ATMS and CSP Techniques. In *Proceedings of the 89 IJCAI Conference* pages 290-296, Detroit, Michigan, 1989.

[9] S. J. Friedman and K. J. Supowtit. Finding the Optimal Oredring for Binary Decision Diagrams. In *Proceedings of the 24th Design Automation Conference,* June 1987.

[10] J. C. Madre and J. P. Billon. Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behaviour. In *Proceedings of the 25th Design Automation Conference,* Anaheim, California, June 1988.

[11] J. C, Madre and O, Coudert. A Complete Reasoning Maintenance System Based On Typed Decision Graphs. Bull Research Report N°90006, March 1990.

[12] L, Maesano. *Specifications fonctionnelles de i'tnterface programmatique a un Systeme de Maintien du Raisonnement* Projet Sachem, June 1989.

[13] S, Minato, N. Ishiura and S. Yajima. Fast Tautology Checking Using Shared Binary Decision Diagrams - Experimental Results. In *Proceedings of the Workshop on Applied Formal Methods for Correct VLSI Design* pages 107-111, Leuven, Belgium, November 1989.

[14] R. Reiter and J. de Kleer. Foundations for Assumption-Based Truth Maintenance Systems. Preliminary Report. In *Proceedings of AAAI-87, American Association for Artificial Intelligence National Conference* pages 183-188, Seattle, July 13-17, 1987,