FOUNDATIONS OF ASSUMPTION-BASED TRUTH MAINTENANCE SYSTEMS: Preliminary Report

Raymond Reiter[1]
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S-1A4

Johan de Kleer
Intelligent Systems Laboratory
XEROX Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

ABSTRACT

In this paper we (1) define the concept of a *Clause Management System* (CMS) — a generalization of de Kleer's ATMS, (2) motivate such systems in terms of efficiency of search and abductive reasoning, and (3) characterize the computation affected by a CMS in terms of the concept of prime implicants.[1]

1. A Problem-Solving Architecture

Figure 1 illustrates an architecture for a problem solving system consisting of a domain dependent Reasoner coupled to a domain independent *Clause Management System* (CMS). For our present purposes, the Reasoner is a black box which, in the process of doing whatever it does, occasionally transmits a propositional clause[2] to the CMS. The Reasoner is also permitted to query the CMS any time it feels so inclined. A query takes the form of a propositional clause C. The CMS is expected to respond with every shortest clause S for which the clause $S \vee C$ is a logical consequence, but S is not a logical consequence, of the clauses thus far transmitted to the CMS by the Reasoner. In Section 2 we show why obtaining such S's is important for many AI systems. For example, for abductive reasoning $\neg S$ will be an hypothesis, which, if known, sanctions the conclusion C. For efficient search $\neg S$ defines a most general context in which C holds.

A traditional ATMS/TMS is a restricted CMS in which (1) the clauses transmitted to the CMS are limited to be either Horn (i.e., justifications) or negative (i.e., nogoods), and (2) the queries (C) are always literals. The fundamental TMS problem is to identify the contexts in which a given singleton clause C holds — this is equivalent to querying the CMS for the shortest clauses S of the preceding paragraph, as the negation of each such S implies C.

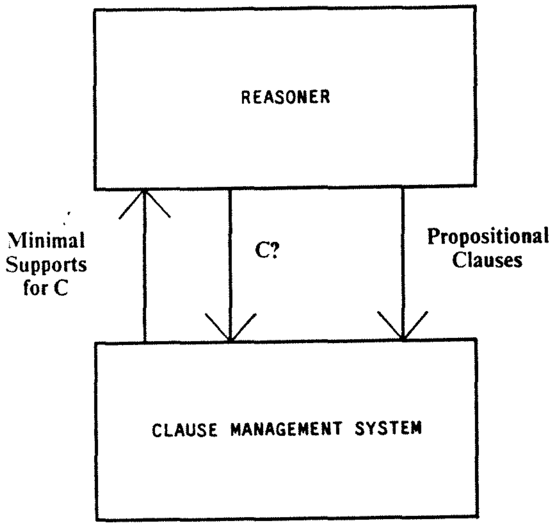

Figure 1 : A problem-solving architecture

2. Motivation and Formal Preliminaries

We shall assume a propositional language with countably infinitely many propositional symbols and with the logical connectives $\vee$, $\neg$. The connectives $\wedge$, $\supset$ are defined in terms of $\vee$, $\neg$ in the usual way, as are the formulas of the language. The definition of the entailment relation, $\models$, is also standard: If S is a set of formulas and w a formula then $S \models w$ just in case every assignment of truth values to the propositional symbols of the language which makes each formula of S true also makes w true.

[1] Fellow of the Canadian Institute for Advanced Reserch. This research was funded by the Canadian National Science and Engineering Research Council under grant A9044.

[2] In actual fact, the reasoner may transmit an arbitrary predicate calculus clause (containing variables for example), but this clause would be treated propositionally by the CMS. In other words, different atomic formulas are treated as different propositional symbols by the CMS.

2.1. Definitions

A *literal* is a propositional symbol or the negation of a propositional symbol. A *clause* is a finite disjunction $L_1 \vee \ldots \vee L_n$ of literals, with no literal repeated. We shall often represent a clause by the set of its literals. The *empty* clause, denoted by $\{\}$ is the clause with no literals. A clause is a *tautology* iff it contains a propositional symbol and the negation of that propositional symbol. Let Σ be a set of clauses, and C a clause. A clause S is a *support* for C with respect to Σ iff $\Sigma \not\models S$ and $\Sigma \models S \cup C$. S is a *minimal support* for C with respect to Σ iff no proper subset of S is a support for C with respect to Σ.

We can think of the CMS as a repository for Σ — some (not necessarily all) of the conclusions derived thus far by the Reasoner[3]. A support clause S for C with respect to Σ has the properties:
1. $\Sigma \models S \cup C$ i.e. $\Sigma \models \neg S \supset C$.
2. $\Sigma \not\models S$ i.e. $\Sigma \cup \{\neg S\}$ is satisfiable.

Property 1 tells us that the conjunction of literals $\neg S$ is an hypothesis which, if known to Σ (and hence to the Reasoner) would sanction the conclusion C. Property 2 precludes hypotheses inconsistent with Σ since these would sanction any conclusion whatsoever. Finally, a minimal support clause S defines a shortest hypothesis $\neg S$ which sanctions C or, as it were, a simplest conjecture from which C follows.

We are now in a position to specify the task which a CMS is to achieve. Recall that a CMS receives clauses transmitted to it by the Reasoner. Let Σ be the set of such clauses. Recall also, that the Reasoner may query the CMS with a clause C. The task of a CMS is to determine all minimal support clauses for C with respect to Σ. Example:

$$\Sigma = \{\{p,q,r\},\{p,\neg q\},\{p,\neg r\},\{\neg p,q,s\},\{q,r,\neg t\}\}$$

Minimal supports for $\{p\}$: $\{\}$.
Minimal supports for $\{\}$: none.
Minimal supports for $\{q\}$: $\{s\},\{r,\neg t\},\{\neg q\}$.
Minimal supports for $\{p,q\}$: $\{\}$
Minimal supports for $\{s,r\}$: $\{q\},\{\neg s\},\{\neg r\}$.

It is important to observe that S being a minimal support clause for C is relative to Σ. In other words, $\neg S$ is a simplest conjecture from which C follows with respect to what the CMS has been told about the knowledge available to the Reasoner. $\neg S$ need not be a simplest conjecture so far as the Reasoner is concerned, since the Reasoner may have information relevant to this question of simplicity which it has failed to transmit to the CMS, or perhaps the Reasoner has failed to derive such relevant information. Why should a Reasoner find this notion of a minimal support clause of any value to it all? There are at least two reasons:

2.2. Abductive Reasoning

Imagine a reasoning system with some knowledge base KB which, for simplicity of exposition, we take to be a set of first order sentences. Imagine further that the Reasoner has some goal formula g which it hopes to establish by a back-chaining inference procedure using KB as its premises, but that these premises are insufficient to prove g. Suppose the Reasoner recognizes this by its inability to expand any of the leaf nodes in the search tree of Figure 2, which we shall use by way of an example[4]. From this the Reasoner can conclude:

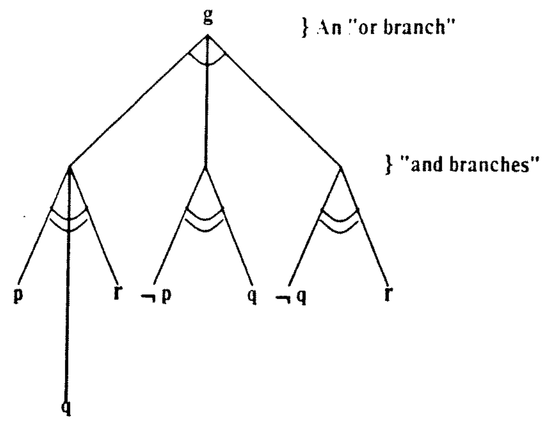

Figure 2 : A back-chaining search tree.

KB $\models p \wedge q \wedge r \supset g$ i.e. KB $\models \neg p \vee \neg q \vee \neg r \vee g$
KB $\models \neg p \wedge q \supset g$ i.e. KB $\models p \vee \neg q \vee g$
KB $\models \neg q \wedge r \supset g$ i.e. $\models q \vee \neg r \vee g$

Now, suppose the reasoner is concerned with performing *abduction*, which is to say that it is seeking an explanation for g. Perhaps g is some observation of the world and KB, the Reasoner's current theory of the world, is inadequate to explain g (i.e., KB $\not\models g$). The explanation which the Reasoner seeks is an hypothesis which, together with its background knowledge KB, entails g. Trivially, for the example of Figure 2, there are three such explanations immediately at hand: $p \wedge q \wedge r$, $\neg p \wedge q$ and $\neg q \wedge r$. But these are not the simplest possible explanations. It is the job of a CMS to provide such simplest explanations. Accordingly, the Reasoner transmits to the CMS the three clauses it inferred from Figure 2. The CMS now contains the set of clauses $\Sigma = \{\{\neg p,\neg q,\neg r,g\},\{p,\neg q,g\},\{q,\neg r,g\}\}$. If the Reasoner now queries the CMS with the clause $\{g\}$ the CMS returns three minimal support clauses of $\{g\}$ with respect to Σ, namely: $\{\neg g\},\{p,\neg q\},\{\neg r\}$. This means

$\Sigma \models g \supset g$ and hence $KB \models g \supset g$,
$\Sigma \models \neg p \wedge q \supset g$ and hence KB $\models \neg p \wedge q \supset g$, and
$\Sigma \models r \supset g$ and hence $KB \models r \supset g$.

Thus, aside from the trivial explanation g, there are two simplest explanations for g, namely $\neg p \wedge q$ and r.

Notice that we have in mind here quite specific notions of "explanation" and "simplest." Explanations are conjunctions of ground literals. A simplest explanation is one for which no proper sub-conjunct is an explanation. Finally, we insist that explanations be consistent with Σ,

[3] Remember that the Reasoner decides, on its own pragmatic grounds, which conclusions it transmits to the CMS and which it withholds.

[4] We are assuming here that g is a ground literal, and that, although KB is a set of first order sentences, the leaf nodes of Figure 2 are all ground literals.

for otherwise we could explain anything!

Notice also that a CMS, as defined, is capable of providing simplest explanations only for g's which are disjunctions of ground literals. This is clearly not as general as one might like. For example, the Reasoner could have two observations g_1 and g_2 of the world for which it wishes simplest explanations i.e., it wishes minimal conjuncts e such that $\Sigma \models e \supset g_1 \wedge g_2$ and $\Sigma \not\models \neg e$. Our CMS is not defined to handle this setting. In the full paper we show how a CMS can.

Finally, with reference to Figure 2, notice that we have taken the Reasoner to generate abductive inferences by a back-chaining mechanism which terminates with leaves of the search tree which cannot be expanded further. While this is one possible mechanism, others are also possible. For example, the Reasoner may have defined some distinguished set of literals which, in a back-chaining search, are never expanded. For the Reasoner, such literals define a class of acceptable assumptions which the Reasoner is prepared to make. Back-chaining is not essential; one can define resolution theorem-provers with suitable termination conditions. The unresolved literals of the uncompleted refutations can support abductive inferences (e.g., [Cox and Pietrzykowski, 1986]). Again, such unresolved literals may be determined by a prespecified class of assumptions acceptable to the Reasoner.

There are many systems and proposals for abductive reasoning along the lines sketched above. Representative examples are residue resolution [Finger, 1985], the THEORIST system of [Poole, 1986], the hypothesis generation formalism of [Cox and Pietrzykowski, 1986], and the NLAG system for learning by analogy by [Greiner, 1986].

2.3. Efficient Search

By exploiting the CMS to organize and control search, much of the computation of the Reasoner can be avoided. Consider the following sequence of statements (from [de Kleer, 86]):

$A : x \in \{0,1\}$
$B : a = e_1(x)$
$C : y \in \{0,1\}$
$D : b = e_2(y)$
$E : z \in \{0,1\}$
$F : c = e_3(z)$
$G : b \neq c$
$H : a \neq b$

The functions e_i require expensive computations, for example, $e_i(x) = (x + 100000)!$.

Suppose that the Reasoner is based on chronological backtracking: it processes the statements A through H one at a time until an inconsistency is detected in which case it backtracks to the most recent variable assignment it can change. The sequence of steps it might follow to find the two solutions are as follows:

1 : Let $x = 0$, compute $a = e_1(0)$.
2 : Let $y = 0$, compute $b = e_2(0)$.
3 : Let $z = 0$, compute $c = e_3(0)$. As $b = c$ backtrack to 3.
4 : Let $z = 1$, compute $c = e_3(1)$, $b \neq c$ but $a = b$ so backtrack to 2.
5 : Let $y = 1$, compute $b = e_2(1)$.
6 : Let $z = 0$, compute $c = e_3(0)$, $b \neq c$, $a \neq b$, solution.
7 : Let $z = 1$, compute $c = e_3(1)$. As $b = c$ backtrack to 1.
8 : Let $x = 1$, compute $a = e_1(1)$.
9 : Let $y = 0$, compute $b = e_2(0)$.
10 : Let $z = 0$, compute $c = e_3(0)$. As $b = c$ backtrack to 10.
11 : Let $z = 1$, compute $c = e_3(1)$, $b \neq c$, $a \neq b$, solution.
12 : Let $y = 1$, compute $b = e_2(1)$.
13 : Let $z = 0$, compute $c = e_3(0)$, $b \neq c$, as $a = b$ backtrack to 13.
14 : Let $z = 1$, compute $c = e_3(1)$, as $b = c$ stop.

Notice that this approach requires 14 expensive computations and 6 backtracks.

Now consider how a CMS might be used to improve this search. The CMS propositional symbols all represent equalities (e.g., '$x = 1$'). The new search is the same as the chronological one with the following changes. Every time the Reasoner does some computation, it constructs a clause representing it (e.g., the computation of $a = e_1(x)$ from $x = 0$ is represented by $x \neq 0 \vee a = e_1(0)$) and conveys this to the CMS. Before performing any computation, the Reasoner checks to determine whether the computation has been done previously. Before choosing (indicated by a 'Let' in the trace) a value for a variable, the Reasoner first queries the CMS to see whether the variable is determined by the current choices. If the variable is determined, no choice is necessary and processing proceeds. If the variable is not determined, it chooses a value which can be consistently added to the current choice set. The resulting problem-solving trace is:

1 : Let $x = 0$, transmit $x = 0 \vee x = 1$, $x \neq 0 \vee a = e_1(0)$.
2 : Let $y = 0$, transmit $y = 0 \vee y = 1$, $y \neq 0 \vee b = e_2(0)$.
3 : Let $z = 0$, transmit $z = 0 \vee z = 1$, $z \neq 0 \vee c = e_3(0)$, $b \neq e_2(0) \vee c \neq e_3(0)$. The current choice set is now inconsistent, so backtrack to 3.
4 : $z = 1$ follows, transmit $a \neq e_1(0) \vee b \neq e_2(0)$. The current choice set is inconsistent, so backtrack to 2.
5 : $y = 1$ follows, transmit $y \neq 1 \vee b = e_2(1)$.
6 : Let $z = 0$, solution.
7 : Let $z = 1$, transmit $z \neq 1 \vee c = e_3(1)$, $b \neq e_2(1) \vee c \neq e_3(1)$. The current choice set is inconsistent, so backtrack to 1.
8 : Let $x = 1$, transmit $x \neq 1 \vee a = e_1(1)$.
9 : Let $y = 0$.
10 : $z = 1$ follows, solution.
11 : Let $y = 1$, transmit $a \neq e_1(1) \vee b \neq e_2(1)$. The current choice set is inconsistent, so stop.

From this example we can see some of the advantages of a CMS-guided search. Intuitively, the CMS is functioning as an intelligent cache. For this example, CMS-approach requires 6, not 14 expensive computations, 3, not 5 backtracks, and 8, not 14 choices. Note that this particular search example exploits only a few of the capabilities of a CMS — we present it only as an illustration of how a CMS could be utilized. It is relatively simple to invent a strategy for this particular problem which achieves the same efficiency, however, the CMS provides a *general* facility that achieves these advantages for any Reasoner.

The CMS performs many of the functions of a conventional TMS [Doyle, 79] [Doyle, 83] [McAllester, 80]. Their advantages (and disadvantages) have been extensively discussed elsewhere (e.g., [de Kleer, 86]).

3. Prime Implicants

Definition. A *prime implicant* of a set Σ of clauses is a clause C such that

1. $\Sigma \models C$, and
2. For no proper subset C' of C does $\Sigma \models C'$.

The concept of a prime implicant arises in solving the problem of two-level Boolean minimization of switching circuits [Birkoff and Bartee 1970, Ch. 6]. In this setting, one is required to synthesize a given Boolean function in sum-of-products form using the fewest total number of and-gates and or-gates. Our definition of prime implicant is the dual of that used in Boolean minimization, basically because for us, the Boolean function is represented by Σ, a set of clauses, and hence is in product-of-sums form. Despite this difference, we shall use the same terminology "prime implicant" since formally both concepts share the same properties modulo the duality between $\vee$ and $\wedge$.

Notice that if $\Sigma \not\models p$ and $\Sigma \not\models \neg p$ for some propositional symbol p, then the tautology $p \vee \neg p$ is a prime implicant of Σ.

The following result is straightforward:

Proposition 1. If Σ is a set of clauses and C a clause, then $\Sigma \models C$ iff there is a prime implicant of Σ which is a subset of C.

Theorem 2. *Suppose Σ is a set of clauses and C a clause. If S is a minimal support clause for C with respect to Σ then there is a prime implicant Π of Σ such that $\Pi \cap C \neq \{\}$ and $S = \Pi - C$.*

Proof. We know that $\Sigma \models S \cup C$. Moreover, by the minimality of S, we know that $S \cap C = \{\}$. By Proposition 1, there is a prime implicant Π of Σ such that $\Pi \subseteq S \cup C$, say $\Pi = S' \cup C'$ where $S' \subseteq S$ and $C' \subseteq C$. We prove first that $C' \neq \{\}$ from which it follows that $\Pi \cap C \neq \{\}$. For if $C' = \{\}$ then $\Pi \subseteq S$ and since $\Sigma \models \Pi$ it must be that $\Sigma \models S$ which contradicts S being a support clause for C with respect to Σ. Finally, we prove that $S' = S$, so that $\Pi = S \cup C'$ and since $S \cap C = \{\}$ and $C' \subseteq C$ it will follow that $S = \Pi - C$. To prove $S' = S$ we assume the contrary and obtain a contradiction. So, suppose S' is a proper subset of S. Since $\Sigma \not\models S$ then $\Sigma \not\models S'$. Moreover, since $\Sigma \models \Pi$ and $C' \subseteq C$ then $\Sigma \models S' \cup C$. But then S' is a smaller support clause for C with respect to Σ than is S, which contradicts the minimality of S. QED.

Unfortunately, the converse of Theorem 2 is false, as the following example shows:

$\Sigma = \{\{p_1, c_1\}, \{p_1, p_2, c_2\}\}$
$C = \{c_1, c_2\}$

The prime implicants of Σ are:

$\{p_1, c_1\}, \{p_1, p_2, c_2\}, \{p_1, \neg p_1\}, \{p_2, \neg p_2\}$, etc.

The prime implicant $\Pi = \{p_1, p_2, c_2\}$ satisfies $\Pi \cap C \neq \{\}$, but $\Pi - C = \{p_1, p_2\}$ is not a minimal support clause for C with respect to Σ.

There is, however, an important partial converse of Theorem 2:

Theorem 3. *Let Σ be a set of clauses and C a non-empty clause. If Π is a prime implicant of Σ such that $C \subseteq \Pi$, then $\Pi - C$ is a minimal support clause for C with respect to Σ.*

Proof. We must prove that $\Sigma \not\models \Pi - C$, which is obvious, and that $\Sigma \models (\Pi - C) \cup C$ which is equally obvious. QED.

Definition. A *unit clause* is a clause with just one literal.

A simple consequence of Theorems 2 and 3 is the following:

Corollary 4. *Let Σ be a set of clauses and $C = \{\ell\}$ a unit clause. Then S is a minimal support clause for C with respect to Σ iff there is a prime implicant Π of Σ such that $\ell \in \Pi$ and $S = \Pi - \{\ell\}$.*

Corollary 4 completely characterizes the minimal support clauses in the case of unit queries issued by the Reasoner to the CMS. As we shall see in Section 5, this result provides a characterization of de Kleer's [1986] Assumption Based Truth Maintenance System. Moreover, it will allow us to generalize his system considerably.

Notation. When Σ is a set of clauses and C a clause,

$\Delta(C, \Sigma) = \{\Pi - C | \Pi$ is a prime implicant of Σ and $\Pi \cap C \neq \{\}\}$

MIN-SUPPORTS$(C, \Sigma) =$

$\{S | S \in \Delta(C, \Sigma)$ and no clause of $\Delta(C, \Sigma)$ is a proper subset of $S\}$.

Theorem 5. *(Characterization of minimal support clauses.) MIN-SUPPORTS (C, Σ) is the set of all minimal support clauses of C with respect to Σ.*

Proof. By Theorem 2, if S is a minimal support clause of C with respect to Σ then $S \in \Delta(C, \Sigma)$. We must prove that no proper subset of S is in $\Delta(C, \Sigma)$. Suppose to the contrary, for some proper subset S' of S, that $S' \in \Delta(C, \Sigma)$. We shall prove that S' is a support clause for C with respect to Σ, contradicting the minimality of S. Clearly, since $\Sigma \not\models S$ and $S' \subseteq S, \Sigma \not\models S'$. It remains to show that $\Sigma \models S' \cup C$. Now $S' = \Pi - C$ for some prime implicant Π. Thus, $S' \cup C = (\Pi - C) \cup C \supseteq \Pi$. Since $\Sigma \models \Pi, \Sigma \models S' \cup C$. Hence, S' is a support clause for C with respect to Σ.

Now suppose $S \in$ MIN-SUPPORTS (C, Σ). We must prove that S is a minimal support clause for C with respect to Σ, i.e., that

1. $\Sigma \not\models S$,
2. $\Sigma \models S \cup C$, and
3. No proper subset of S has properties 1 and 2.

Proof of 1:

Since $S = \Pi - C$ for some prime implicant Π of Σ such that $\Pi \cap C \neq \{\}$, S is a proper subset of Π. Because Π is a prime implicant of $\Sigma, \Sigma \not\models S$.

Proof of 2:

Since $S = \Pi - C$ for some prime implicant Π, $S \cup C = (\Pi - C) \cup C \supseteq \Pi$. Since $\Sigma \models \Pi, \Sigma \models S \cup C$.

Proof of 3:

Assume to the contrary that S has a proper subset S' with property 2, i.e., that $\Sigma \models S' \cup C$. By Proposition 1, Σ has a prime implicant $\Pi' \subseteq S' \cup C$. Since $S = \Pi - C$ for some prime implicant Π of $\Sigma, S \cap C = \{\}$. Since $S' \subseteq S$, $S' \cap C = \{\}$. Hence, since $\Pi \subseteq S' \cup C, \Pi - C \subseteq S'$ which is a proper subset of S; since $\Pi' - C \in \Delta(C, \Sigma)$, $S \notin$ MIN-SUPPORTS (C, Σ), contradiction. QED.

4. Interpreted vs. Compiled

There are two natural ways the CMS can store information and process queries issued to it by the Reasoner.

4.1 The Interpreted Approach

The simplest storage mechanism is to encode the Reasoner's clauses just as they are, possibly indexed by the literals they contain for more efficient content addressable access. Thus, updating the CMS's database with a new clause is quick and simple. The price one pays for this simplicity of storage is a high retrieval cost. To find all minimal support clauses for C with respect to Σ, the CMS's database requires computing MIN-SUPPORTS (C, Σ) by Theorem 5, and this can be an expensive computation.[5] If the Reasoner is expected to issue many CMS updates but few queries, then this interpreted approach will be warranted. In the full paper we shall describe and justify an algorithm for computing MIN-SUPPORTS (C, Σ).

4.2 The Compiled Approach

Under this approach, the CMS does not store the clauses transmitted to it by the Reasoner. Instead, it stores all the prime implicants of these clauses. This is potentially an explosive approach. It can be shown that there are Boolean functions in n variables with exponentially (in n) many prime implicants. Moreover, CMS updates can be very expensive since, if Σ is the CMS's current database (consisting of the prime implicants of all clauses issued by the Reasoner thus far), and K is a new clause issued by the Reasoner, we must compute all the prime implicants of $\Sigma \cup \{K\}$. The reward for the high space and time complexity of this approach, by Theorem 5, is that retrieval of minimal support clauses is cheap.

[5] In fact, it is easy to show that the general problem is NP-hard.

The first thing we must show is that there is no loss of information in representing a set Σ of clauses by $PI(\Sigma)$ the set of its prime implicants, i.e., that Σ and $PI(\Sigma)$ are logically equivalent.

Theorem 6. *Suppose Σ is a set of clauses. Then Σ and $PI(\Sigma)$ are logically equivalent in the sense that if $C \in \Sigma$, then $PI(\Sigma) \models C$, and if $C \in PI(\Sigma)$ then $\Sigma \models C$.*

Proof. Trivial.

Theorem 6 justifies the compiled approach of storing only the prime implicants of the Reasoner's clauses in the CMS database.

In the full paper we shall describe and justify an algorithm for updating a compiled CMS database i.e., for computing the prime implicants of $\Sigma \cup \{K\}$ assuming we already have all prime implicants of Σ.

5. De Kleer's ATMS: A Reconstruction

De Kleer's [1986] Assumption-Based Truth Maintenance System (ATMS) is a CMS constrained to process so-called Horn clauses. Moreover, the ATMS requires that the propositional symbols have a distinguished subset called assumptions. From the standpoint of the Reasoner, an assumption might be one of the distinguished propositional symbols which it is prepared to propose as part of an hypothesis to explain an observation in abductive reasoning (Section 2), or one of the propositional symbols forming part of a proposed solution to a constraint satisfaction problem (Section 2).

Definition. A *Horn* clause is a clause in which at most one propositional symbol occurs unnegated.

The general form of a Horn Clause is $\neg p_1 \vee \cdots \vee \neg p_n \vee p$ for propositional symbols $p, p_1, \cdots p_n$, $n \geq 0$, or $\neg p_1 \vee \cdots \vee \neg p_n, n \geq 0$.

Recall that, for the purposes of de Kleer's ATMS, there is a distinguished subset of the propositional symbols called assumptions. We denote assumptions by upper-case A's, usually subscripted, non-assumption propositional symbols by lower case p's, and when the distinction is unimportant by lower-case α's.

In de Kleer's approach, the Reasoner is constrained to transmit to the ATMS only Horn clauses. De Kleer calls such transmitted Horn clauses *justifications*. When a clause has the form $\neg\alpha_1 \vee \cdots \vee \neg\alpha_n \vee \alpha$, α is called the *consequent* of the clause, and $\alpha_1, \cdots \alpha_n$ the *antecedents* of the clause. If $n = 0$, the consequence α is called a *premise*. When formulated in our terms, the task of the ATMS is the following:

Given J, the set of justifications transmitted thus far to the ATMS by the Reasoner, and α, a propositional symbol (which may or may not be an assumption), compute $\{A_1 \wedge \cdots \wedge A_n | \{\neg A_1, \cdots, \neg A_n\}$ is a minimal support clause for $\{\alpha\}$ with respect to $J\}$.

This set is what de Kleer calls a consistent, sound, complete and minimal label for α. Corollary 4 immediately provides the following:

Theorem 7. *(Characterization of de Kleer's ATMS) Suppose that J is the set of justifications transmitted to the ATMS by the Reasoner, and that $\{\alpha\}$ is a query, where α is a propositional symbol (which may or may not be an assumption). Then the answers to this query are given by $\{A_1 \wedge \cdots \wedge A_k | k \geq 0$ and $\neg A_1 \vee \cdots \vee \neg A_k \vee \alpha$ is a prime implicant of $J\}$.*

In the full paper we characterize the algorithm used by de Kleer's ATMS, and prove its correctness with respect to Theorem 7.

6. Generalizing the ATMS

We can immediately see various ways to generalize de Kleer's ATMS. To begin, justifications need not be Horn clauses. Thus we can define a justification to be any clause of the form

$\pm\alpha_1 \vee \cdots \vee \pm\alpha_n \vee \alpha$, where $n \geq 0$ and each α is a propositional symbol which may or may not be an assumption.

Moreover, the consequence α need not be atomic. We can allow $\neg\alpha$ as a consequence, or more generally, $\pm\alpha_1 \vee \cdots \vee \pm\alpha_k$ can be taken to be a consequence. Finally, queries can be arbitrary clauses, not necessarily, as in de Kleer's ATMS, unit clauses. In the full paper, we elaborate on such possible generalizations. Notice that Theorem 5 characterizes query evaluation for any such generalization.

7. A Word on Computing Prime Implicants

The results of this paper rely on computing all, or some, prime implicants of set Σ of propositional clauses. In the theory of switching circuit Boolean minimization, prime implicants are computed using the consensus method [Birkhoff and Bartee, 1970, Ch. 6]. Since our notion of prime implicant is the dual of that for switching theory, we are concerned with the dual of the *consensus* method, which turns out to be resolution [Robinson, 1965]. A brute force way of computing all prime implicants of Σ is to resolve pairs of clauses of Σ, add the resolvents to Σ, delete subsumed clauses and repeat until no fresh clauses are obtained. The resulting clauses are all of the prime implicants of Σ. Obviously, we prefer a more disciplined approach to computing prime implicants. There are a few such approaches in the literature, e.g., [Minicozzi and Reiter, 1972] [Slagle *et al.*, 1969]). The full paper considers the appropriateness of these and other algorithms for determining prime implicants.

References

[Birkhoff and Bartee, 1970] G. Birkhoff and T.C. Bartee. *Modern Applied Algebra.* McGraw-Hill, New York, 1970.

[Cox and Pietrzykowski, 1986] P.T. Cox and T. Pietrzykowski. Causes for events: their computation and applications. In *Proc. 8th Int. Conf. on Autom. Deduction* and *Lecture Notes in Computer Science 230*, pages 608–621, Springer-Verlag, 1986.

[de Kleer, 1986] J. de Kleer. An assumption-based TMS. *Artificial Intelligence* **28** 127–162, 1968.

[Doyle, 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence* **12** 231–272, 1979.

[Doyle, 1083] J. Doyle. *Some theories of reasoned assumptions: An essay in rational psychology.* CS-83-125, Department of Computer Science, C.M.U., 1983.

[Finger, 1985] J.J. Finger. *Residue: a deductive approach to design synthesis.* Technical Report Stan-CS-85-1035, Knowledge Systems Laboratory, Stanford University, 1985.

[Greiner, 1986] R. Greiner. *Learning by understanding analogies.* Technical Report CSRI-188, Department of Computer Science, University of Toronto, 1986.

[McAllester, 1980] D. McAllester. *An outlook on truth maintenance.* AIM-551, Artificial Intelligence Laboratory, M.I.T., 1980.

[Minicozzi and Reiter, 1972] E. Minicozzi and R. Reiter. A note on linear resolution strategies in consequence-finding. *Artificial Intelligence* **3** 175–180, 1972.

[Poole, 1986] D. Poole. *Default reasoning and diagnosis as theory formation.* Department of Computer Science Techical Report CS-86-08, University of Waterloo, 1986.

[Robinson, 1965] J.A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM* **12** 23–41, 1965.

[Slagle *et al.*, 1969] J.R. Slagle, C.L. Chang, and R.C.T. Lee, Completeness theorems for semantic resolution in consequence-finding. In *Proceedings IJCAI-69*, pages 281–285, Washington, D.C., 1969.