DIMACS Technical Report 94-42
August 1994

Essential and Redundant Rules in Horn Knowledge Bases[a]

by

Peter L. Hammer[b] **and Alexander Kogan**[b,c]

[a]The authors gratefully acknowledge the partial support of AFOSR (Grant F49620-93-1-0041) and ONR (Grants N00014-92-J1375 and N00014-92-J4083).
[b]RUTCOR, Rutgers University, P.O. Box 5062, New Brunswick, NJ 08903-5062
e-mail: hammer@rutcor.rutgers.edu
[c]Accounting and Information Systems, Faculty of Management, Rutgers University, 180 University Ave., Newark, NJ 07102, e-mail: kogan@rutcor.rutgers.edu

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

A production rule of a knowledge base is called *essential* if it is present in any prime knowledge base which is logically equivalent to the given one. Identification of essential rules constitutes a crucial part of the structural analysis of any knowledge base. It specifies the degree of freedom we have in constructing logically equivalent transformations of the base, by specifying the set of rules which must remain in place, and implicitly showing which rules could be replaced by other ones.

A prime rule is called *redundant* if it is not present in any irredundant prime knowledge base which is logically equivalent to the given one. Redundant rules can be viewed as the "least important" prime rules of a knowledge base. The recognition of redundancy of a particular prime rule in a knowledge base will eliminate such rule from consideration in any future simplifications of the knowledge base.

This paper provides combinatorial characterization of essential and redundant rules of propositional Horn knowledge bases, and develops, whenever possible, efficient computational procedures for recognizing essentiality and redundancy of prime rules of Horn knowledge bases.

1 Introduction

Expert systems represent one of the most rapidly developing areas of decision support systems. Production rule based systems account for the vast majority of expert systems used in practice. The simplest type of production rule based systems are the propositional ones. A propositional production rule base consists of a set of implications of the type

if		the propositional variable x is *true*
	and	the propositional variable y is *true*
	and	the propositional variable z is *true*
then		the propositional variable w is *true*,

and it is usually written as

$$xyz \to w.$$

With this notation, a production rule base can be written as

$$\left.\begin{array}{llllllllllll} uvw & \to & x, & x & \to & u, & x & \to & v, & x & \to & w, \\ uvw & \to & y, & y & \to & u, & y & \to & v, & y & \to & w, \\ uvw & \to & z, & z & \to & u, & z & \to & v, & z & \to & w. \end{array}\right\} \quad (1)$$

Let $\neg x$ denote the "negation" of x, i.e. the propositional variable which is true if and only if the propositional variable x is false. Propositional variables, as well as their negations, will be called *literals*. Non-negated variables will be called *positive* literals, while negated variables will be called *negative* literals. We shall denote the value of a propositional variable by 1 (TRUE) or 0 (FALSE), and a truth value assignment to all the propositional variables will be represented as a 0-1 vector.

Using the standard terminology of artificial intelligence, we shall call *models* those 0-1 vectors for which all the implications in the rule base hold; we shall call *non-models* those 0-1 vectors which violate at least one of the implications. For example, for the rule base (1), with propositional variables x, y, z, u, v, w, the vector $(0, 0, 0, 0, 1, 1)$ is a model, while the vector $(0, 1, 1, 1, 1, 1)$ is a non-model since it violates the first rule of (1).

It is important to note that the same set of models can be defined by various different rule bases. In such case we shall call the corresponding rule bases *logically equivalent*, or simply *equivalent*. This terminology is most natural since the essential role of a rule base is simply to define the set of models. It can be verified that the rule base (1) is logically equivalent to the substantially simpler rule base

$$\left.\begin{array}{lllllllll} uvw & \to & x, & x & \to & y, & y & \to & z, \\ z & \to & u, & z & \to & v, & z & \to & w. \end{array}\right\} \quad (2)$$

Substituting the rule base (2) for the rule base (1) "compresses" the knowledge represented by (1), thus reducing the computer memory requirements and accelerating answering queries.

A rule base can be viewed as a mechanism which determines whether a given 0-1 vector is or is not a model. It is natural therefore to associate with a rule base the following Boolean function

$$f(\alpha) = \begin{cases} 1, & \alpha \text{ is a model} \\ 0, & \text{otherwise} \end{cases}$$

where α is a 0-1 vector having as its length the number of distinct propositional variables in the rule base.

In order to make the translation of rule base concepts to Boolean ones more workable, we shall use the representation of production rules as clauses, based on the fact that a production rule $\bigwedge_{i=1}^{n} q_i \rightarrow \bigvee_{j=1}^{m} r_j$ holds if and only if the clause $(\bigvee_{i=1}^{n} \neg q_i) \vee (\bigvee_{j=1}^{m} r_j)$ is true.

As an example, the production rules

$$\begin{array}{lcl} uv & \rightarrow & x, \\ wv & \rightarrow & x, \\ wzu & \rightarrow & x, \end{array}$$

can be represented as the clauses

$$\begin{array}{l} \neg u \vee \neg v \vee x, \\ \neg w \vee \neg v \vee x, \\ \neg w \vee \neg z \vee \neg u \vee x, \end{array}$$

respectively. Therefore, the base consisting of the above three production rules is equivalent to the conjunction of the above three clauses:

$$(\neg u \vee \neg v \vee x)(\neg w \vee \neg v \vee x)(\neg w \vee \neg z \vee \neg u \vee x).$$

Reasoning as in this example, it can be seen that every production rule base can be represented as a *conjunctive normal form* (CNF). This notation is more convenient for logically equivalent transformations of rule bases.

A very important way of simplifying knowledge bases is built around the concept of *primality*. Consider, for example, the CNF

$$(x \vee y \vee \neg z)(y \vee z)(\neg y \vee \neg z) \tag{3}$$

A systematic examination of (3) shows that if we drop the literal $\neg z$ in the first clause, the resulting CNF

$$(x \vee y)(y \vee z)(\neg y \vee \neg z) \tag{4}$$

will have exactly the same models as (3).

In order to describe such simplification process in precise terms, we shall need the concept of an *implicate*. For this purpose, let us consider the Boolean function defined by the CNF (3). It is easy to notice that if $f = 1$, then $x \vee y \vee \neg z = 1$. A clause C with the property that

$$(f = 1) \Rightarrow (C = 1)$$

is called an *implicate* of f. An implicate C is called *prime* iff for any clause C' obtained from C by deleting a literal, the implication

$$(f = 1) \Rightarrow (C' = 1)$$

is false. For instance, $x \vee y \vee \neg z$ in the example above is not prime, since the clause $x \vee y$ obtained by deleting $\neg z$, is still an implicate. However, deleting any of the literals in $x \vee y$ produces a clause which is not an implicate of f; hence, $x \vee y$ is a prime implicate of f.

Clearly, non-prime clauses in a CNF can be replaced by shorter ones, and this process can be repeated as long as the CNF containes non-prime clauses.

A CNF representing a function is called *prime* iff each clause of the CNF is a prime implicate of the function. It can be seen that the CNF (4) is prime, while the logically equivalent CNF (3) is not.

A transformation of a given CNF to an equivalent prime one will simplify the structure of a knowledge base. Unfortunately, algorithms transforming an arbitrary CNF to a prime one are computationally difficult (the problem is NP-hard). However, for the classes of CNFs having the major role in artificial intelligence (i.e. *Horn formulae*) these computations can be carried out in a very efficient way.

We are not going to deal in this paper with methods of obtaining prime implicates. Perhaps, the best known method for the solution of this problem is the so-called *consensus* or *resolution* method, see [1, 15, 16]. It is known that finding all the prime impicates of a Boolean function is in general difficult from a computational point of view, although efficient methods are available for certain classes of problems.

A classical result in Boolean algebra states that any Boolean function can be expressed as the conjunction of *all* its prime implicates. For example, the Boolean function f given by the CNF (4) has a total of 4 prime implicates, and can be written as their conjunction:

$$f = (x \vee y)(y \vee z)(\neg y \vee \neg z)(x \vee \neg z) \tag{5}$$

Prime implicates are the building blocks of any Boolean function. The conjunction of all of them always represents the function, but this representation may be "redundant", as in the example above: representation (4) of f shows that the last clause in the representation (5) is not needed and can be dropped without changing the function. This shows that some prime implicates of a Boolean function may be "more important" than the others. The concept of *essential* implicates formalizes the intuitive notion of the "most important" prime implicates.

A prime implicate of a Boolean function is called *essential* if it is contained in every prime CNF of the function. For example, it can be shown that the last two clauses in the CNF (4) are essential, while the clauses $x \vee y$ and $x \vee \neg z$ are not. Essential implicates can be viewed as the "core" of a Boolean function in the sense that their conjunction must be present in every prime CNF of the function.

Identification of essential rules constitutes a crucial part of the structural analysis of any knowledge base. It specifies the degree of freedom we have in constructing logically equivalent transformation of the base, by specifying the set of rules which must remain in place, and implicitly showing which rules could be replaced by other ones.

Another aspect refelecting the relative importance of different prime implicates is formalized in the concept of *redundancy.* To introduce this we need the notion of an *irredundant* CNF. A CNF representing a function is called *irredundant* iff dropping any clause from it produces a CNF which does not represent the same function.

For illustration, let us consider the CNF

$$(x \vee \neg y)(y \vee \neg z)(x \vee \neg z) \tag{6}$$

It is easy to notice that if $x \vee \neg y$ is true, and if $y \vee \neg z$ is true, then $x \vee \neg z$ is also true. It is clear then that (6) is true if and only if

$$(x \vee \neg y)(y \vee \neg z) \tag{7}$$

is true. Hence, the clause $x \vee \neg z$ in (6) can be dropped without changing the function represented by (6). It is easy to see that the CNF (7) is irredundant. It can also be seen that this CNF is prime.

Transforming a given production rule base to a prime and irredundant one is a natural way of simplifying the struture of knowledge. It is therefore natural to relate the importance of prime implicates to their participation in irredundant and prime CNFs.

A prime implicate of a Boolean function is called *redundant* if no irredundant prime CNF representing the function contains it. It can be seen that the CNF (7) is the only irredundant and prime CNF of the function f it represents, and that $x \vee \neg z$ is a prime implicate of f. Therefore, $x \vee \neg z$ is a redundant implicate of f.

Redundant implicates can be viewed as the "least important" prime implicates of a Boolean function. They can not participate in any economical representation of a given knowledge. Therefore, recognizing that a particular prime rule in a knowledge base is redundant will eliminate such rule from consideration in any future simplifications of the knowledge base.

2 Horn Functions

Although various aspects of inference and knowledge compression are computationally difficult, many real life applications can be handled efficiently due to their specific structure. The most widely used and well known example is *Horn* production rule bases where inference has linear time complexity, see [3].

A production rule is called *Horn* iff it is of the form

$$\bigwedge_{x \in S} x \rightarrow y$$

or

$$\bigwedge_{x \in S} x \rightarrow \text{FALSE}$$

(with the usual convention that $\bigwedge_{x\in\emptyset} x = 1$). The specific feature of these rules consist of the fact that no negative literals appear on the left hand side of the implication, and that at most one literal appears on the right hand side of it. The equivalent clauses will be of the form

$$y \vee \bigvee_{x\in S} \neg x \tag{8}$$

or

$$\bigvee_{x\in S} \neg x, \tag{9}$$

i.e. they will contain at most one positive literal. Clauses with this property will be called *Horn* clauses. In order to distinguish between these two types of clauses, we shall call those of the form (8) *definite Horn* clauses, and those of the form (9) *negative* clauses.

Conjunctive normal forms consisting only of Horn clauses will be called *Horn* CNFs. Those CNFs that involve only definite Horn clauses will be called *definite Horn* CNFs, and those that involve only negative clauses will be called *negative* CNFs. For example, the CNF $(x \vee \neg y)(\neg y \vee \neg z)$ is Horn, while the CNF $(x \vee y)(\neg y \vee \neg z)$ is not Horn. The CNF $(x \vee \neg y)(y \vee \neg z)$ is definite Horn, and the CNF $(\neg x \vee \neg y)(\neg y \vee \neg z)$ is negative.

It is clear from the discussion in the previous section that any CNF defines a Boolean function, but a Boolean function can be defined by many different CNFs. If at least one of the CNFs representing a Boolean function is Horn, we shall call the Boolean function itself *Horn*. For example, the Boolean function

$$f = (x \vee \neg y \vee \neg u)(\neg z \vee \neg u \vee w)(\neg u \vee \neg w) \tag{10}$$

is Horn, since the CNF representing it is Horn. It should however be noted that the same function f can be represented by a non-Horn CNF; e.g.

$$f = (x \vee \neg y \vee \neg u \vee z)(\neg z \vee \neg u \vee w)(\neg u \vee \neg w). \tag{11}$$

The justification of the term *Horn function* lies in the following

Proposition 2.1 (see [5]) *Each prime implicate of a Horn function is either definite Horn or negative.*

Therefore, every prime CNF of a Horn function is Horn. It turns out that an irredundant prime representation of a Horn function can be obtained easily:

Proposition 2.2 (see [5]) *Any Horn CNF can be transformed to an equivalent irredundant prime CNF in time quadratic in the length of the given CNF.*

3 Decomposition of Horn Functions

Let us first remind that a *unit* clause is a clause consisting of exactly one literal. It is easy to see that any unit prime implicate of a Boolean function is essential. Moreover, if x or $\neg x$ is a unit prime implicate of a Boolean function f, then no other prime implicate of f depends on the variable x. Hence, any Horn function f (given by a Horn CNF) can be decomposed in quadratic time into a (possibly empty) conjunction of literals g, and a Horn function f' having no unit prime implicates, such that g and f' depend on disjoint sets of variables, and $f = g \bigwedge f'$. Therefore, **without loss of generality, from now on we shall restrict our attention to Horn functions which do not have unit prime implicates.**

Given a Horn function f, the set of its definite Horn prime implicates and the set of its negative prime implicates will be denoted by $\mathcal{H}(f)$ and $\mathcal{N}(f)$ respectively. We shall make use below of the CNF $H(f) = \bigwedge_{C \in \mathcal{H}(f)} C$.

Any Horn CNF can be obviously written as a conjunction of a negative CNF and a definite Horn one. However, the Boolean functions defined by the two parts obtained in this way may depend on the original CNF rather than be characteristic of the underlying Boolean function. Consider, for example, the two Horn CNFs

$$(\neg u \vee \neg v \vee \neg x)(\neg w \vee \neg y \vee z)(\neg x \vee y)(x \vee \neg y)(\neg x \vee z)(x \vee \neg z) \tag{12}$$

and

$$(\neg u \vee \neg v \vee \neg z)(\neg w \vee \neg x)(\neg x \vee y)(\neg y \vee z)(x \vee \neg z) \tag{13}$$

representing the same function. The definite Horn parts of them

$$(\neg w \vee \neg y \vee z)(\neg x \vee y)(x \vee \neg y)(\neg x \vee z)(x \vee \neg z) \tag{14}$$

and

$$(\neg x \vee y)(\neg y \vee z)(x \vee \neg z) \tag{15}$$

represent different Horn functions. Similarly, the negative parts of (12) and (13)

$$(\neg u \vee \neg v \vee \neg x)$$

and

$$(\neg u \vee \neg v \vee \neg z)(\neg w \vee \neg x)$$

also define different functions.

In order to avoid this ambiguity, we shall define the *definite Horn component* h_f of a Horn function f as a function represented by the conjunction of all definite clauses of an arbitrary prime CNF of f. This definition is consistent, as shown by

Proposition 3.1 (see [5]) *Let f be a Horn function and*

$$F = H \wedge G \tag{16}$$

be a prime CNF of it, where H is the conjunction of all definite Horn clauses in (16), and G is the conjunction of all negative clauses in (16). If

$$F' = H' \wedge G' \tag{17}$$

is another prime Horn CNF of f (H' and G' being defined in a way similar to (16)), then H' and H are equivalent.

Let us define a *negative restriction* of a Horn function f as being the conjunction of the negative clauses appearing in an irredundant and prime CNF of f. It is then natural to ask whether results similar to Proposition 3.1 hold for the various negative restrictions of f. Consider, for example, the prime and irredundant CNF

$$(\neg u \vee \neg v \vee \neg x)(\neg w \vee \neg y)(\neg x \vee y)(\neg y \vee z)(x \vee \neg z), \tag{18}$$

representing the same Horn function as (13). It is easy to see that, although (13) and (18) represent the same function, the conjunction of the negative clauses of (13)

$$(\neg u \vee \neg v \vee \neg z)(\neg w \vee \neg x)$$

and of (18)

$$(\neg u \vee \neg v \vee \neg x)(\neg w \vee \neg y)$$

represent different Boolean functions. However, it can be proved that

Proposition 3.2 (see [5]) *All the negative restrictions of a Horn function contain the same number of clauses.*

In order to obtain a structural description of the negative restrictions of a Horn function f, we construct a directed graph $G_N(f)$ in the following way. The vertices of $G_N(f)$ correspond to the negative prime implicates of f. The vertices corresponding to two prime implicates C_1 and C_2 are linked by an arc $C_1 \to C_2$ if C_2 is an implicate of $C_1 \wedge H(f)$. It follows from the definition that $G_N(f)$ is transitively closed, and hence each strongly connected component of $G_N(f)$ is a complete directed subgraph.

Let $\mathcal{N}_i(f)$, $i \in \{1, \ldots, k\}$, be the subsets of negative prime implicates of f corresponding to the strongly connected components of $G_N(f)$, indexed in such a way that the "starting components", i.e. those with no incoming arcs are at the beginning of the list; let m be the number of starting components.

Proposition 3.3 (see [5]) *A negative conjunctive normal form $\mathcal{F}_N$ is a negative restriction of a Horn function f if and only if it has the form $\mathcal{F}_N = \bigwedge_{i=1}^{m} C_i$, where C_i is a clause in $\mathcal{N}_i(f)$.*

The structure of irredundant prime Horn CNFs is characterized by

Theorem 3.4 (Decomposition Theorem, see [5]) *A conjunctive normal form F is an irredundant and prime representation of a Horn function f if and only if*

$$F = H_f \wedge F^-$$

where H_f is an irredundant prime CNF of the definite Horn component $h(f)$, and where F^- is a negative restriction of f.

4 Forward Chaining Procedure

Given a definite Horn CNF F, we shall define a *forward chaining* procedure which associates with any subset S of variables a superset $\mathcal{R}$ of it in the following way. The procedure takes as input the subset S of variables, initializes the set $\mathcal{R} = S$, and at each step it looks for a definite Horn clause $x' \vee \bigvee_{x \in S'} \neg x$ in $\mathcal{F}$ such that $S' \subseteq \mathcal{R}$, and $x' \notin \mathcal{R}$. If such a clause is found, the variable x' is included into $\mathcal{R}$, and the search is repeated as many times as possible. The final set $\mathcal{R}$ will be denoted S^F and will be called the forward chaining closure of S under F. Different variants of the forward chaining procedure are commonly used as built-in inference strategies in expert systems (see e.g. [8]).

The following two statements proved in [6] characterize the implicates of an arbitrary Horn CNF in terms of the forward chaining procedure.

Lemma 4.1 *A definite Horn clause $C = x' \vee \bigvee_{x \in S} \neg x$ is an implicate of a definite Horn function f given by a Horn CNF F if and only if $x' \in S^F$.*

It follows that the closure S^F depends only on the function f, and not on a particular Horn CNF of it. Therefore, we may also denote the same set as S^f. Clearly, if $S_2 \subseteq S_1^f$, then $S_2^f \subseteq S_1^f$.

Lemma 4.2 *A negative clause $C_1 = \bigvee_{x \in S_1} \neg x$ is an implicate of a Horn function $f = F \wedge C_2$, where F is a definite Horn CNF and $C_2 = \bigvee_{x \in S_2} \neg x$ is a negative clause, if and only if $S_2 \subseteq S_1^F$.*

The following statement proved in [7] shows that the forward chaining procedure can be efficiently implemented. We assume that the starting subset of variables S contains only variables present in the given definite Horn CNF F.

Lemma 4.3 *Given a definite Horn CNF F and a subset of variables S, the forward chaining procedure for F starting with S can be performed in time linear in the length of F.*

5 Main Results

Now we are ready to study systematically essential and redundant prime implicates of a Horn function. Let us denote by $\mathcal{N}_0(f)$ the set of all those negative prime implicates of a Horn function f which do not belong to any starting component $\mathcal{N}_i(f)$, $i = 1, \ldots, m$.

Proposition 3.3 and Theorem 3.4 immediately imply

Proposition 5.1 *The set of redundant negative prime implicates of a Horn function f is $\mathcal{N}_0(f)$.*

This structural characterization does not provide directly an efficient computational procedure, since the number of negative prime implicates of a Horn function f can be exponential in the length of a CNF representation of f. A computationally efficient algorithm for checking the redundancy of a negative prime implicate of a Horn function is given in the proof of the following statement.

Theorem 5.2 *Given a negative prime implicate C of a Horn function f represented by an irredundant prime CNF $F = H_f \wedge F^-$, where H_f is a definite Horn CNF and F^- is a negative CNF, it can be checked in $O(pL)$ time whether C is a redundant prime implicate of f; here L is the length of F and p is the number of clauses in F^-.*

Proof The procedure consists of checking for every clause $C' \in F^-$ whether C' is an implicate of $H_f \vee C$. Since for every clause C this can be verified in time $O(L)$, the procedure can be accomplished in $O(pL)$ time. It follows from the definition of the sets $\mathcal{N}_i(f)$ and from Proposition 3.3 that the clause C belongs to the same set $\mathcal{N}_i(f)$ as C' if and only if this condition holds. Therefore, by Proposition 5.1 the clause C is redundant if and only if there is no such clause C' in F^-. □

As an example, consider the Horn function represented by the following CNF

$$F = (\neg x \vee y)(\neg y \vee \neg z)(\neg y \vee \neg w).$$

It can be checked that F is irredundant and prime, and that $\neg x \vee \neg z$ is a prime implicate of F. Since

$$\{x, z\} \not\subseteq \{y, z\}^F = \{y, z\} \quad \text{and} \quad \{x, z\} \not\subseteq \{y, w\}^F = \{y, w\},$$

the prime implicate $\neg x \vee \neg z$ is redundant. It can be shown in the same way that the prime implicate $\neg x \vee \neg w$ is also redundant.

Proposition 3.3 and Theorem 3.4 immediately imply

Proposition 5.3 *A negative clause C is an essential prime implicate of a Horn function f if and only if C is the only clause in its set $\mathcal{N}_i(f)$.*

A computationally efficient algorithm for checking the essentiality of a negative prime implicate of a Horn function is given in the proof of the following

Theorem 5.4 *Let F be an irredundant prime Horn CNF of length L representing a Horn function f, and let C be a negative clause of length q in F; it can be checked in $O(qL)$ time whether C is an essential prime implicate of f.*

Proof Let $C = \bigvee_{x \in S} \neg x$ and $F = H \wedge F^-$, where H is a definite Horn CNF and F^- is a negative CNF. By Lemma 4.2, a clause $C_* = \bigvee_{x \in S_*} \neg x$ is an implicate of $F = H \wedge C$ if and only if $S \subseteq S_*^H$, and vice versa C is an implicate of $F = H \wedge C_*$ if and only if $S_* \subseteq S^H$. If C and C_* are distinct prime implicates, then $S \setminus S_* \neq \emptyset$. Therefore, the set $\mathcal{N}_i(f)$ containing C will contain at least one more clause if and only if there is $x' \in S$ such that the clause $\bigvee_{x \in S^H \setminus \{x'\}} \neg x$ is an implicate of $C \wedge H$, where S^H is the forward chaining closure of S under H. For every $x' \in S$ this condition can be checked in linear time. □

For illustration, consider the Horn function represented by the following CNF

$$F = (\neg x \vee \neg y \vee z)(\neg z \vee x)(\neg z \vee y)(\neg z \vee \neg w)(\neg w \vee \neg u).$$

It can be checked that F is irredundant and prime. Let H denote the conjunction of definite Horn clauses in F. Since

$$\{z, w\}^F = \{z, w, x, y\} \quad \text{and} \quad \{z, w\} \subseteq \{w, x, y\}^H,$$

the prime implicate $\neg z \vee \neg w$ is not essential. On the other hand, since

$$\{w,u\}^F = \{w,u\}, \quad \{w,u\} \not\subseteq \{w\}^F = \{w\}, \quad \text{and} \quad \{w,u\} \not\subseteq \{u\}^F = \{u\},$$

the prime implicate $\neg w \vee \neg u$ is essential. An irredundant prime CNF equivalent to F is

$$(\neg x \vee \neg y \vee z)(\neg z \vee x)(\neg z \vee y)(\neg x \vee \neg y \vee \neg w)(\neg w \vee \neg u).$$

It follows from Theorem 3.4 that a definite Horn prime implicate C is an essential (redundant) implicate of a Horn function f if and only if C is an essential (redundant) implicate of its definite Horn component $h(f)$. A redundant definite Horn prime implicate of f is a redundant definite Horn prime implicate of $h(f)$, while a redundant definite Horn prime implicate of $h(f)$ is either a redundant definite Horn prime implicate of f, or it is a non-prime implicate of f. Therefore, without loss of generality, we will restrict our attention below to the analysis of essentiality (redundancy) of definite Horn prime implicates of definite Horn functions only.

A computationally efficient algorithm for checking the essentiality of a prime implicate of a definite Horn function is given in the proof of the following statement.

Theorem 5.5 *Let F be an irredundant prime Horn CNF of length L representing a definite Horn function f, and let C be a clause of length q in F; it can be checked in $O(qL)$ time whether C is an essential prime implicate of f.*

Proof A prime implicate $C = y \vee \bigvee_{x \in S} \neg x$ is not essential if and only if it is implied by $H(f) \setminus \{C\}$, i.e. $y \in S^{H(f)\setminus\{C\}}$. Let $C' = y \vee \bigvee_{x \in S'} \neg x$ be the clause used by the forward chaining procedure to include y into the set $\mathcal{R}$. Clearly, for any $x' \in S' \setminus S$, the clause $x' \vee \bigvee_{x \in S} \neg x$ is an implicate of f. Therefore, $S' \subseteq S^F \setminus \{y\}$. Since C and C' are distinct prime implicates, $S \setminus S' \neq \emptyset$. Therefore, to check whether C is an essential prime implicate of f it is sufficient to perform the following steps:

- Use the forward chaining procedure to obtain the set S^F in linear time.
- For every variable $x' \in S$ the forward chaining procedure is used to check in linear time whether $y \vee \bigvee_{x \in S^F \setminus \{y,x'\}} \neg x$ is an implicate of f.

The clause C is essential if and only if there is no variable $x' \in S$ such that $y \vee \bigvee_{x \in S^F \setminus \{y,x'\}} \neg x$ is an implicate of f. $\square$

For illustration, let us consider a definite Horn function represented by the following CNF:

$$F = (\neg x \vee y)(\neg x \vee z)(\neg y \vee u)(\neg z \vee w)(\neg u \vee \neg w \vee x).$$

It can be checked that F is irredundant and prime. Using the forward chaining procedure we get that $\{x\}^F = \{x, y, z, u, w\}$. Since both $\neg z \vee \neg u \vee \neg w \vee x$ and $\neg y \vee \neg u \vee \neg w \vee x$ are implicates, neither $\neg x \vee y$ nor $\neg x \vee z$ is essential. Since $\{u,w\}^F = \{u, w, x, y, z\}$, and since both $\neg u \vee \neg y \vee \neg z \vee x$ and $\neg w \vee \neg y \vee \neg z \vee x$ are implicates, it follows that $\neg u \vee \neg w \vee x$ is not essential either. On the other hand, since $\{y\}^F = \{y, u\}$ and u is not an implicate, it follows that $\neg y \vee u$ is essential. Similarly, $\neg z \vee w$ is essential as well. An irredundant prime CNF equivalent to F is

$$(\neg y \vee u)(\neg z \vee w)(\neg u \vee \neg w \vee y)(\neg u \vee \neg w \vee z)(\neg y \vee \neg z \vee x)(\neg x \vee u)(\neg x \vee w).$$

We will show below that it is computationally difficult to check whether a prime implicate of a definite Horn function is redundant. More specifically, we will analyze the computational complexity of

The Definite Horn Redundant Implicate Problem

Instance: A definite Horn CNF F of function h and a prime implicate C of f.

Question: Is C a non-redundant implicate of h?

It can be seen easily that this problem belongs to NP. If the answer is **yes** and a Horn CNF F' containing C is given, it can be checked in quadratic time that F' is an irredundant prime CNF of h. Remark that F' can be only polynomially longer than F (see [6]).

Theorem 5.6 *The Definite Horn Redundant Implicate Problem is NP-complete.*

Proof We will show how the following problem can be reduced in polynomial time to the Definite Horn Redundant Implicate Problem.

The Prime Attribute Name Problem

Instance: A definite Horn CNF H in variables $x_1, \ldots, x_n$.

Question: Is there a negative prime implicate of $H \wedge (\bigvee_{i=1}^{n} \neg x_i)$ containing x_1?

It has been shown in [10] that the Prime Attribute Name Problem is NP-complete. An instance of the Prime Attribute Name Problem can be transformed to an instance of the Definite Horn Redundant Implicate Problem in the following way. Let us consider the definite Horn CNF

$$F = H \wedge \bigwedge_{i=1}^{n} (\neg t \vee x_i),$$

where t is a new variable. Since F has no unit prime implicates, each $\neg t \vee x_i$ is prime. We will prove that $\neg t \vee x_1$ is a non-redundant prime implicate if and only if there is a negative prime implicate of $H \wedge (\bigvee_{i=1}^{n} \neg x_i)$ containing x_1. To see this, it is sufficient to prove that F' is an irredundant prime CNF equivalent to F if and only if

$$F' = H' \wedge \bigwedge_{x \in S} (\neg t \vee x),$$

where H' is an irredundant prime CNF equivalent to H, and $\bigvee_{x \in S} \neg x$ is a prime implicate of $H \wedge (\bigvee_{i=1}^{n} \neg x_i)$. Clearly, there are no prime implicates of F' containing non-negated t. Therefore, according to Lemma 4.1 H' has to be an irredundant prime CNF equivalent to

H. Then Lemmas 4.1 and 4.2 imply that $\bigvee_{x \in S} \neg x$ is an implicate of $H \wedge (\bigvee_{i=1}^{n} \neg x_i)$ if and only if $H' \wedge \bigwedge_{x \in S}(\neg t \vee x)$ is equivalent to F. □

The last proof implies that it is computationally difficult to check redundancy even when the prime implicate in question is quadratic.

Clearly, any clause in an arbitrary irredundant prime CNF is a non-redundant implicate of the function. This gives a trivial necessary condition of redundancy. Two polynomially verifiable sufficient conditions of redundancy are presented below.

Let E_h denote the conjunction of all essential prime implicates of a definite Horn function h. It follows from Theorem 5.5 that the CNF E_h can be constructed in time quadratic in the length of a given Horn CNF of h. This gives a quadratic time sufficient condition of redundancy using the following

Proposition 5.7 *A prime implicate C of a Boolean function f is redundant if it is implied by the conjunction of its essential prime implicates E_f.*

If it happens that all the clauses in an irredundant prime CNF are essential, then any other prime implicate of the function is redundant, and this CNF is the unique irredundant prime CNF of the function. Acyclic Horn knowledge bases (see [7]) provide an example of such functions. However, there are Horn functions in this class which are not acyclic. Consider, for example, the definite Horn CNF

$$F = (\neg x \vee y)(\neg y \vee \neg z \vee w)(\neg w \vee y)(\neg w \vee z).$$

It can be checked that this CNF is irredundant and prime, and that each clause of it is essential. This CNF and the definite Horn function it represents are not acyclic. Nevertheless, as follows from the above, F is the unique irredundant and prime CNF of the function, and all the other prime implicates are redundant. This function has only one additional prime implicate $\neg x \vee \neg z \vee w$, which is redundant.

As another illustration, let us consider a definite Horn function represented by the following irredundant prime CNF:

$$F = (\neg x \vee y)(\neg y \vee z)(\neg u \vee w)(\neg w \vee u)(\neg z \vee \neg w \vee t). \tag{19}$$

Using Theorem 5.5, it can be seen that all the clauses in F, except the last one, are essential. Their conjunction implies the prime implicate $\neg x \vee z$, showing that it is redundant. On the other hand, the prime implicates $\neg y \vee \neg w \vee t$, $\neg x \vee \neg w \vee t$, $\neg y \vee \neg u \vee t$, and $\neg x \vee \neg u \vee t$ are not implied by that conjunction, in spite of the fact that all of them are redundant. We will develop below another polynomially verifiable sufficient condition of redundancy that could be used to show that these prime implicates are redundant.

The development that follows is based on the concept of *direct determination* introduced in [11]. Given a definite Horn CNF F, we shall call two subsets of variables S_1 and S_2 equivalent under F if $S_2^F = S_1^F$. Clearly, the equivalence of two given subsets of variables can be checked in linear time. It also follows from Lemma 4.1 that this equivalence depends only on the definite Horn function f represented by F, not on a particular CNF representation of f. Therefore, we shall say that S_1 and S_2 are equivalent under f.

An implicate $C_* = y \vee \bigvee_{x \in S_*} \neg x$ of a definite Horn function f is called *direct* if there is a subset $\mathcal{H}$ of implicates of f such that C_* is an implicate of $\bigwedge_{C \in \mathcal{H}} C$ and for every $C = x' \vee \bigvee_{x \in S} \neg x$ in $\mathcal{H}$ the set S is not equivalent to the set S_* under f.

This definition may suggest that to check the directness of an implicate $C_* = y \vee \bigvee_{x \in S_*} \neg x$ we should generate all the implicates of f, then select all implicates $C = x' \vee \bigvee_{x \in S} \neg x$ such that S and S_* are not equivalent, and then check whether C_* is an implicate of their conjunction. This approach is clearly intractable, since the number of implicates is typically exponentially large. The following statement, which is a reformulation of the result proved in [11], shows that the directness of an implicate can be easily seen in any Horn CNF of the function.

Lemma 5.8 *If $C_* = y \vee \bigvee_{x \in S_*} \neg x$ is a direct implicate of a definite Horn function f, if F is an arbitrary Horn CNF of f, and if G is the conjunction of all clauses $C = x' \vee \bigvee_{x \in S} \neg x$ of F such that S and S_* are not equivalent, then C_* is implied by G.*

Proof Since C_* is a direct implicate, there must exist an inclusionwise minimal set of implicates $\mathcal{H}$ such that C_* is an implicate of $H = \bigwedge_{C \in \mathcal{H}} C$, for every $C = z \vee \bigvee_{x \in S} \neg x$ in $\mathcal{H}$ the set S is not equivalent to the set S_* under f, and for any $C' \in \mathcal{H}$ the CNF $\bigwedge_{C \in \mathcal{H} \setminus \{C'\}} C$ does not imply C_*. These conditions mean that the forward chaining procedure for H starting with S_* will use each clause of H to include y into the set $\mathcal{R}$. Therefore, for every clause $C = z \vee \bigvee_{x \in S} \neg x$ in $\mathcal{H}$ we have $S \subseteq S_*^f$.

Let F be an arbitrary Horn CNF of f and let G be the conjunction of all clauses $C = z \vee \bigvee_{x \in S} \neg x$ in F such that S is not equivalent to S_* under f. If we prove that each $C' \in \mathcal{H}$ is an implicate of G, this would mean that C_* is an implicate of G as well, thus proving the lemma. Let us assume that there is a clause $C_1 = x_1 \vee \bigvee_{x \in S_1} \neg x$ in $\mathcal{H}$ which is not an implicate of G. Since C' is an implicate of f, the forward chaining procedure for F starting with S_1 must include x_1 into $\mathcal{R}$ using a clause $C_2 = x_2 \vee \bigvee_{x \in S_2} \neg x$ such that S_2 and S_* are equivalent. Therefore, $S_2 \subseteq S_1^f$, and since $S_* \subseteq S_2^f$, we have $S_* \subseteq S_1^f$. But, as we saw in the previous paragraph, $S_1 \subseteq S_*^f$. Therefore, S_* and S_1 are equivalent, contradicting the assumptions of our construction. □

Lemma 5.8 implies that to check whether $C_* = y \vee \bigvee_{x \in S_*} \neg x$ is a direct implicate of a definite Horn CNF F it is sufficient to use the forward chaining procedure to obtain the set S_*^F and the sets S^F for all the clauses $C = z \vee \bigvee_{x \in S} \neg x$ in F. The forward chaining procedure can be used then to check whether C_* is an implicate of $\bigwedge_{C \in \mathcal{H}} C$; here $\mathcal{H}$ is the set of all the clauses $C = z \vee \bigvee_{x \in S} \neg x$ in F such that S is not equivalent to S_* under F. This procedure takes only $O(pL)$ time, where p is the number of clauses in F and L is the length of F.

Theorem 5.9 *If a prime implicate of a definite Horn function is direct, then it is redundant.*

Proof Let us assume that a direct prime implicate $C_* = y \vee \bigvee_{x \in S_*} \neg x$ is not redundant. Then there must exist an irredundant prime CNF F of f containing C_*. Let $\mathcal{H}$ be the set of all the clauses $C = z \vee \bigvee_{x \in S} \neg x$ in F such that S is not equivalent to S_* under f. Clearly, $\mathcal{H}$ does not contain C_*. By Lemma 5.8, C_* is implied by $\bigwedge_{C \in \mathcal{H}} C$. Therefore, C_* can be dropped from F without changing the function, and the CNF F is not irredundant, contradicting our assumption. $\Box$

It follows from the above that directness is an easily verifiable sufficient condition of redundancy. As an example, let us consider the CNF (19). It can be checked that $\{x\}^F = \{x, y, z\}$, $\{y\}^F = \{y, z\}$, $\{z\}^F = \{z\}$, $\{u\}^F = \{u, w\}$, $\{w\}^F = \{u, w\}$, $\{z, w\}^F = \{z, u, w, t\}$. Now let us consider the prime implicate $\neg y \vee \neg w \vee t$. Since $\{y, w\}^F = \{y, z, u, w, t\}$, there is no clause in F whose negative part is equivalent to $\{y, w\}$. Therefore, this prime implicate is direct, and hence redundant. It can be seen in the same way that the prime implicates $\neg x \vee \neg w \vee t$, $\neg y \vee \neg u \vee t$, and $\neg x \vee \neg u \vee t$ are also redundant.

References

[1] A. Blake. *Canonical Expression in Boolean Algebra*. Dissertation, Chicago, 1937.

[2] E. Charles and O. Dubois. MELODIA: Logical methods for checking knowledge bases. In: *Validation, Verification and Test of Knowledge-based Systems*, Eds.: M. Ayel and J.-P. Laurent, Wiley & Sons 1991, 95–105.

[3] W.F. Dowling and J.H. Gallier. Linear time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, **3** (1984), 267–284.

[4] A. Ginsberg. Knowledge-base reduction: A new approach to checking knowledge bases for inconsistency & redundancy. *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, 1988, 585–589.

[5] P.L. Hammer and A. Kogan. Horn functions and their DNFs. *Information Processing Letters*, **44** (1992), 23–29.

[6] P.L. Hammer and A. Kogan. Optimal compression of propositional Horn knowledge bases: complexity and approximation. *Artificial Intelligence*, **64** (1993), 131–145.

[7] P.L. Hammer and A. Kogan. *Quasi-Acyclic Propositional Horn Knowledge Bases: Optimal Compression*. RUTCOR Research Report RRR 10-93, Rutgers University, New Brunswick, NJ, July 1993.

[8] J.P. Ignizio. *Introduction to Expert Systems: The Development and Implementation of Rule-Based Expert Systems*. McGraw-Hill, New York, 1991.

[9] H.R. Lewis. Renaming a set of clauses as a Horn set. *Journal of the ACM*, **25** (1978), 134–135.

[10] C.L. Lucchesi and S.L. Osborn. Candidate keys for relations. *Journal of Computer and System Sciences*, **17** (1978), 270–279.

[11] D. Maier. Minimum covers in the relational database model. *Journal of the ACM*, **27** (4) (1980), 664–674.

[12] M. Minoux. LTUR: A simplified linear-time unit resolution algorithm for Horn formulae and computer implementation. *Information Processing Letters*, **29** (1988), 1–12.

[13] M. Minoux. The Unique Horn-Satisfiability problem and quadratic Boolean equations. *Annals of Mathematics and Artificial Intelligence*, **6** (1992), 253–266.

[14] T.A. Nguyen, W.A. Perkins, T.J. Laffey, D. Pecora. Knowledge base verification. *AI Magazine*, Summer 1987, Vol. 8, No. 2, 69–75.

[15] W. Quine. A way to simplify truth functions. *American Mathematical Monthly*, **62** (1955), 627–631.

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

[17] J. Tepandi. Comparison of expert system verification criteria: redundancy. In: *Validation, Verification and Test of Knowledge-based Systems*, Eds.: M. Ayel and J.-P. Laurent, Wiley & Sons 1991, 49–62.