Consequence Finding Algorithms

Pierre Marquis
CRIL / Université d'Artois
rue de l'Université – S.P. 16
F–62307 Lens Cedex – France
E-mail: marquis@cril.univ-artois.fr

Abstract

This chapter is about consequence finding algorithms, i.e., procedures for deriving theorems that are intensionally characterized within a logical knowledge base. Such theorems can be those built up from a preset language, those arising when some additional information is incorporated into the knowledge base, etc. Consequence finding is investigated in propositional logic. Various computational techniques are presented and the complexity of consequence finding is discussed. The role of consequence finding for assumption-based reasoning (including abduction and simple default reasoning) and for circumscription and related forms of closed world reasoning is also exhibited.

1 Introduction

In this section, the notion of consequence finding is introduced and motivated in informal terms. Then, the scope of the chapter and its organization are successively pointed out.

1.1 Motivations

Consequence finding algorithms are procedures for deriving extensionally some specific consequences of a logical knowledge base that are characterized in an intensional way. The notion of consequence finding proves helpful in the various situations where querying a knowledge base does not simply consist in *checking* whether a given formula can be derived from the base (the proof finding situation) but in *finding out* some theorems of interest; these theorems are typically not given in an explicit way but only characterized by some properties they must satisfy. The set of formulas characterized by such properties is a subset of the representation logical language often called a *production field* (or, in a more transparent way, a *target language*); the logical consequences of the knowledge base belonging to this set are called its *characteristic theorems*. Thus, in simple words, consequence finding is oriented towards the derivation of characteristic theorems from a knowledge base given a production field.

Interestingly, many production fields can be considered; they correspond to the various possible ways to grasp the contents of a knowledge base and/or to focus on some parts of it. Thus, a knowledge engineer facing a consistent knowledge base can be interested in exhibiting the facts that can be soundly derived from it, then the clauses of size 2, and so on. This may help him understanding what really is in the base, starting with the logically strongest pieces of knowledge. Accordingly, limiting the size of the consequences of interest gives rise to a simple family of production fields. Because such production fields contain only polynomially many clauses (in the propositional case), computing all the logically strongest "short" clauses of a knowledge base (i.e., those of size lower than a given bound) leads to a polynomially-sized approximation of the knowledge base (which is in the general case logically weaker than the original knowledge base, but can nevertheless be used to answer all the "short" queries in an efficient, sound and complete way). This kind of approximation seems to be the best we can hope for in knowledge compilation. Much more sophisticated production fields can be considered. Thus, when querying a database, a front-end user is not always able to formulate exactly what she is looking for; in this situation, giving her in a structured way all the theorems that are about the topics of interest that appear in her queries can be helpful. Various notions of aboutness can be used here. Especially, one can be interested in deriving only those characteristic theorems that are built up from a preset language. Production fields may also relate to the dynamics of the knowledge base. Thus, when pieces of information are incorporated in an incremental way within a knowledge base, one can be interested in evaluating the influence of each incoming formula over the whole knowledge base by focusing on the new theorems that can be generated by expanding the base with the formula. In this situation, the production field is the set of formulas that are *not* theorems of the original base, while they are theorems of the expanded one.

Obviously, consequence finding can be considered as an extension of proof finding (or query answering in a limited sense) since the latter can be recovered from the former by considering a production field restricted to a given formula. However, the variety of production fields that can be taken into account actually makes consequence finding much more powerful than proof finding w.r.t. query answering, even in the case where queries are given in an explicit way. Especially, several forms of common sense reasoning (that can be used to define sophisticated query answering) can be characterized through consequence finding. Thus, beyond determining whether or not a given query is a logical consequence of a knowledge base, consequence finding can help determining why (by pointing out one of the logically strongest theorems implying the query) or why not (by deriving the logically weakest theorems that would enable the query being a consequence of the base, would such theorems be added to it). Indeed, characteristic theorems may capture abductive explanations for queries w.r.t. a knowledge base. Through abduction, consequence finding also proves helpful for circumscription and related forms of closed world reasoning, allowing non-monotonic interpretations of a knowledge base to be achieved.

1.2 The scope of the chapter

The purpose of this chapter is to provide the reader with a comprehensive presentation of the notion of consequence finding, and to show the role it plays for common sense reasoning. To be more precise, we aim at providing one with materials for addressing the following issues:

- How to compute characteristic theorems w.r.t. several families of production fields and what is the corresponding computational complexity?
- How various forms of common sense reasoning, especially abduction, simple forms of default reasoning, circumscription and closed world reasoning can be characterized through consequence finding?

For the sake of simplicity and homogeneity, we shall mainly focus on consequence finding in propositional logic. Especially, every knowledge base under consideration will be represented as a finite set of propositional formulas. Some of the results presented in this chapter can nevertheless be lifted to some restricted fragments of first-order logic and we shall mention it when it is the case.

Four families of production fields are to be considered in the following, namely those that prove particularly helpful for common sense reasoning. The first production field we shall focus on is the whole set of propositional clauses, the second one the set of all clauses built up from a given set of literals, the third one the set of all clauses that are not consequences of some formula (typically, a knowledge base before expansion), and finally, the fourth one is the intersection of the second and the third ones, i.e., the set of all clauses built up from a given set of literals and are not consequences of a given formula. For each of these families, the logically strongest characteristic theorems are among the logically strongest clausal consequences of the knowledge base, i.e., its *prime implicates*. We shall introduce three additional notions of prime implicates so as to capture the notions of logically strongest characteristic clauses w.r.t. the three remaining families of production fields.

As we shall see in what follows, consequence finding mainly consists in deriving such prime implicates or formulas equivalent to their conjunction. Subsequently, in order to address the first issue listed above, the stress will be laid on prime implicates and their computation.

So as to address the second issue, we shall show how several forms of common sense reasoning can be characterized through consequence finding. Especially, we shall describe how characteristic theorems can be used to capture abductive explanations. Such explanations can be exploited to achieve simple default reasoning and to characterize circumscription and related forms of closed world reasoning. In particular, determining whether or not a given formula is a logical consequence of a circumscribed knowledge base amounts to determining whether or not some characteristic clauses exist. Implementations of common sense reasoning are often based on assumption-based truth maintenance systems (ATMS) and/or its extensions (clause management systems (CMS), assumption-based clause management systems (ACMS)). We shall show that such systems actually implement consequence finding (and are based on some properties of prime implicates that are reported in the following).

The great majority of the results presented in the chapter are not original ones. We did not try to trace back in a systematic way those related to prime implicates because the spectrum of domains where prime implicates/implicants have been used is too wide; especially, some algorithms for computing prime implicates/implicants have been rediscovered, sometimes with slight variations, during the past four decades. For similar causes, our presentation of prime implicates is not exhaustive (the number of algorithms proposed so far for computing prime implicates is too large for describing them all in few pages [1]); instead, the focus is laid on the metatheoretic properties of prime implicates that serve as theoretical grounds for many prime implicates algorithms. Accordingly, our purpose mainly is to present the essential features of prime implicates in an organized way, centered around their uses for consequence finding.

1.3 The organization of the chapter

The rest of this chapter is organized as follows. In Section 2, some formal preliminaries are put forward. The next sections are the core of the chapter. Section 3 is devoted to the computational aspects of consequence finding. Especially, the four notions of prime implicate we shall focus on are pointed out. For each notion of prime implicate, some metatheoretic properties are presented, especially those that can be used to compute prime implicates; some complexity issues are also discussed. Section 4 is about consequence finding for common sense reasoning. Several forms of common sense reasoning (mainly abduction, simple default reasoning, circumscription and closed world reasoning) are defined and characterized through consequence finding. Finally, Section 5 concludes this chapter. Some proofs are sketched in an appendix; due to space limitations, most of them are accessible through bibliographical references, only.

[1] The reader interested in a historical perspective of algorithms for computing prime implicants can look at the very nice introduction of [Strzemecki, 1992].

2 Formal Preliminaries

In this section, some formal preliminaries about propositional logic and computational complexity are successively presented.

2.1 Propositional logic

Let PS denote a denumerable set of *propositional symbols* (also called *variables* or *atoms*). For every subset V of PS, $PROP_V$ denotes the *propositional language* built up from the symbols of PS, the boolean constants $true$ and $false$ and the connectives $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\oplus$ in the standard way. The elements of $PROP_{PS}$ are called *formulas*. The *size* $|\Sigma|$ of a formula Σ is the number of symbols (including the connectives and the boolean constants) used to write it. Among formulas are propositional symbols, literals, clauses and terms.

Every propositional symbol of PS is also called a *positive literal* and a negated one a *negative literal*. A *literal* is either a *standard literal*, i.e., a positive literal v or a negative literal $\neg v$ where v is a symbol of PS, or a boolean constant (possibly negated). The boolean constants are viewed as positive literals, and their negations as negative literals. For every literal l, l and its negation $\neg l$ are said *complementary literals*. For every subset V of PS, L_V (resp. L_V^+, L_V^-) is the set of literals (resp. positive literals, negative literals) built up from the propositional symbols of V. Thus, $L_V^+ = V \cup \{true, false\}$ and $L_V^- = \{\neg v; v \in L_V^+\}$. For every set of literals $L = \{l_1, ..., l_n\}$, $\neg L$ denotes the set of negated literals from L: $\neg L = \{\neg l_1, ..., \neg l_n\}$.

Every finite disjunction of literals is called a *clause* and every finite conjunction of literals is called a *term* (sometimes referred to as a *cube*). The empty disjunction of literals is called the *empty clause*, noted $\Box$. Every clause containing only one literal is called a *unit clause*. A clause γ is *reduced* iff it contains only $true$ as a literal, or it does not contain any occurrence of the boolean constant $false$ and any variable occurring in it occurs only once.

For every formula Σ, $\Sigma_{v \leftarrow true}$ (resp. $\Sigma_{v \leftarrow false}$) denotes the formula obtained by substituting in a uniform way every occurrence of the propositional symbol v occurring in Σ by the boolean constant $true$ (resp. $false$). $Var(\Sigma)$ denotes the set of propositional variables occurring in Σ.

Formulas are interpreted in the classical way. An *interpretation* (over $PROP_{PS}$) is a mapping I that associates every propositional symbol to one of the two truth values of $BOOL = \{0, 1\}$ and $I(true) = 1$ and $I(false) = 0$. The semantics $[\![\Sigma]\!]$ of a formula Σ is a boolean function: a truth value can be associated to Σ as long as an interpretation I is considered. $[\![\Sigma]\!](I)$ denotes the truth value taken by Σ within I; it is defined in the usual compositional way. When I is such that $[\![\Sigma]\!](I) = 1$, I is said a *model* of Σ, noted $I \models \Sigma$. Otherwise, I is a *counter-model* of Σ. When a formula has a model, it is said *satisfiable* (or, equivalently, consistent). When a formula has no model (resp. no counter-model), it is said *contradictory* (or, equivalently, inconsistent or unsatisfiable) (resp. *valid*). Valid clauses are also referred to as *tautologies*. The binary relation $\models$ over $PROP_{PS}$ is defined by $\Sigma \models \Phi$ iff every model of Σ is a model of Φ. $\models$ is referred to as *logical deduction*: whenever $\Sigma \models \Phi$ holds, Φ is said a *logical consequence* of Σ, or a Σ-*consequence* for short. Σ is also said *logically stronger* than Φ. The *deductive closure* of Σ is the set of all formulas that are logical consequences of Σ. Whenever $\Sigma \models \Phi$ and $\Phi \models \Sigma$ both hold, Σ and Φ are said *logically equivalent*, noted $\Sigma \equiv \Phi$. More generally, whenever $(\Sigma \wedge \Phi) \equiv (\Sigma \wedge \Psi)$ holds, Φ and Ψ are said Σ-*equivalent*, noted $\Phi \equiv_\Sigma \Psi$. Clearly enough, $\equiv$ and $\equiv_\Sigma$ are equivalence relations over $PROP_{PS}$.

It is easy to realize that every non-reduced clause γ from $PROP_{PS}$ can be turned in an efficient way into an equivalent reduced clause. To do it, it is sufficient to replace every occurrence of $\neg true$ (resp. $\neg false$) by $false$ (resp. $true$), then check whether γ contains $true$ or a pair of complementary literals. If it is the case, γ is rewritten into $true$. Otherwise, every occurrence of $false$, and every occurrence (except one) of every redundant literal are removed from it. At the end, a reduced clause equivalent to γ is obtained. Such a rewriting can be achieved in time polynomial in $|\gamma|$.

Every set of formulas is interpreted conjunctively; especially, when it is finite, a set of formulas is also identified with the formula that is the conjunction of its elements. The cardinal number of a set S is denoted $card(S)$. A *conjunctive* (resp. *disjunctive*) formula is the conjunction (resp. disjunction) of some formulas. It is well-known that every formula admits a conjunction of clauses (resp. a disjunction of terms) equivalent to it; such a conjunction of clauses (resp. disjunction of terms) is called a *conjunctive normal form* (CNF for short) (resp. a disjunctive normal form (DNF for short)) of the formula.

A finite set of terms is a *reduced* DNF of a formula Σ iff:

- It is $\{false\}$ if Σ is unsatisfiable,
- Otherwise, it is a DNF of Σ s.t.
  - No occurrences of the boolean constants $true$ and $false$ occur in it, and
  - Every literal occurs at most once in every term of this set, and
  - No unsatisfiable terms occur in it.

Every DNF formula Σ can be turned in an efficient way into an equivalent reduced DNF. To do it, it is sufficient to successively rewrite every occurrence of $\neg true$ (resp. $\neg false$) by $false$ (resp. $true$), then remove $true$ from every term and remove every term containing $false$, remove in every term every occurrence (except one) of every redundant literal and remove every redundant term, remove unsatisfiable terms and return $\{false\}$ whenever the resulting set of terms is empty. Such a reduction process can be easily achieved in time polynomial in the size of the original DNF Σ.

Both conjunctive normal form and disjunctive normal form are specific cases of so-called *negative normal form*: a formula is said in negative normal form (NNF for short) where the only binary connectives occurring in it are $\wedge$ and $\vee$, the scope of each occurrence of the negation connective $\neg$ is reduced to a propositional symbol, and no occurrence of the boolean constants $true$, $false$ occur in it. Interestingly, every formula Σ can be tranformed into an equivalent NNF formula in polynomial time (as long as it does not contain any occurrence of the connectives $\Leftrightarrow$ and $\oplus$). Indeed, it is sufficient to first remove useless boolean constants (using the standard boolean laws), then rewrite every subformula of the form $\alpha \Rightarrow \beta$ occurring in Σ into $\neg\alpha \vee \beta$, then push down every occurrence of the negation connective $\neg$ within the resulting formula, replacing every binary connective $\wedge$ (resp. $\vee$) by the dual one $\vee$ (resp. $\wedge$) and applying involution of the negation connective (every subformula of the form $\neg\neg\alpha$ is rewritten into α), until the scope of every negation connective is a variable. Loosely speaking, we shall call the resulting formula the NNF of Σ. Contrariwise, a formula cannot always be normalized into CNF (or DNF) in polynomial time (except when some additional symbols are introduced but in this case, logical equivalence is ensured over the original language, only: the initial formula and its normalization share the same logical consequences over the original language but not necessarily over the enriched one). For every formula Σ, $Lit(\Sigma)$ denotes the set of standard literals occurring in the NNF of Σ. In particular, when Σ is a clause, $Lit(\Sigma)$ is the set of standard literals occurring in it.

A CNF formula is *Horn* (resp. *reverse Horn*) iff every clause in it contains at most one standard positive (resp. negative) literal. A CNF formula Σ is *renamable Horn* iff there exists a subset L of $Lit(\Sigma)$ s.t. the formula obtained by substituting in Σ every literal l of L by its negation $\neg l$ is a Horn formula. A *Krom* formula is a CNF formula in which every clause is binary, i.e., contains at most two standard literals. Interestingly, Horn, reverse Horn, renamable Horn, and Krom formulas are *tractable* for the satisfiability problem SAT, i.e., for each class of formulas, there exists a polynomial time algorithm for determining whether a CNF formula belongs to that class, and there exists a polynomial time algorithm for determining whether any formula from the class is satisfiable or not.

2.2 Computational complexity

One of the main purposes of complexity theory[2] is to classify problems according to their requirements on computational resources depending on

[2] See e.g., [Papadimitriou, 1994] for details.

the size of the input. In this framework, a problem is a generic question, i.e., a set of specific instances. Specifically, a *decision problem* is one that has only "yes" and "no" as possible answers. Formally, a decision problem can be considered as a formal language consisting of the set of all its "yes" instances.

An *efficient algorithm* (resp. an efficient transformation) is one that run (resp. can be achieved) in time polynomial in the size of the input (assuming a standard model of computation, e.g., a sequential deterministic Turing machine). Accordingly, a decision problem is considered *efficiently solvable* iff there exists an algorithm that can classify every instance of it in a number of computational steps that is polynomially bounded.

The class of all decision problems that are solvable in polynomial time is denoted by P. All the remaining ones require exponential time and are considered not efficiently solvable. In order to determine whether a problem is efficiently solvable or not, it is sufficient to point out an efficient algorithm to solve it, or to prove that such an algorithm is impossible.

Unfortunately, for many problems, no polynomial algorithms are known but it seems to be impossible to prove that super-polynomial time is actually required. For all these problems, the separation efficiently solvable vs. non-efficiently solvable is not fine-grained enough to classify them in a computationally valuable way: there is need for more refined classes for problems that are not known in P.

The notion of non-deterministic Turing machine is an important tool to achieve this goal. Thus, the class of all languages (encoding decision problems) that can be recognized in polynomial time by such a non-deterministic machine is denoted by NP. Because a deterministic Turing machine can be considered as a non-deterministic one, the inclusion $\mathsf{P} \subseteq \mathsf{NP}$ is etablished; however, the converse is the famous open problem: $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ (that is conjectured false). Among all the problems in NP, the hardest ones are those into which every problem in NP can be *polynomially many-one reduced*: such problems are referred to as NP-*complete*. If any of them has a polynomial algorithm, then $\mathsf{P} = \mathsf{NP}$ holds. Accordingly, it is believed that it is impossible to solve NP-complete problems in polynomial time. SAT, the problem of determining whether a propositional formula in conjunctive normal form is satisfiable, is the prototypical NP-complete problem. Its complementary problem $UNSAT$ (consisting of determining whether a propositional formula in conjunctive normal form is unsatisfiable) is not necessarily in NP (in contrast to P, NP is not known to be closed under complementation). It is assigned to the class coNP that contains the complementary problems to problems of NP. It is conjectured that $\mathsf{NP} \neq \mathsf{coNP}$.

To go further into the classification of non-efficiently solvable problems, another important tool is the notion of Turing machine (deterministic or non-deterministic) *with oracle*. Let X be a class of decision problems. P^X

(resp. NP^X) is the class of all decision problems that can be solved in polynomial time using a deterministic (resp. non-deterministic) Turing machine that can use an oracle for deciding any language of X for "free" (i.e., within a constant, unit time). On this basis, the classes Δ_k^p, Σ_k^p and Π_k^p can be defined by:

- $\Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathsf{P}$,
- $\Delta_{k+1}^p = \mathsf{P}^{\Sigma_k^p}$, with $k \geq 0$,
- $\Sigma_{k+1}^p = \mathsf{NP}^{\Sigma_k^p}$, with $k \geq 0$,
- $\Pi_{k+1}^p = \mathsf{co}\Sigma_{k+1}^p$, with $k \geq 0$.

Thus, $\Sigma_1^p = \mathsf{NP}$ and $\Pi_1^p = \mathsf{coNP}$. The *polynomial hierarchy* PH is the union of all Σ_k^p (for k integer). While it is easy to check that both $\Delta_k^p \subseteq \Sigma_{k+1}^p$, $\Delta_k^p \subseteq \Pi_{k+1}^p$, $\Sigma_k^p \subseteq \Delta_{k+1}^p$, and $\Pi_k^p \subseteq \Delta_{k+1}^p$ hold for every k, it is unknown whether the inclusions are proper or not (but it is conjectured that it is the case). Thus, it is strongly believed that the polynomial hierarchy does not collapse, i.e., is a truly infinite hierarchy.

In order to discriminate further among the problems from $\Delta_2^p = \mathsf{P}^{\mathsf{NP}}$, one can focus on the number of calls to an NP-oracle that are used. Thus, the classes BH_k and coBH_k are defined in the following way:

- $\mathsf{BH}_0 = \mathsf{P}$,
- $\mathsf{BH}_1 = \mathsf{NP}$,
- $\mathsf{BH}_2 = \{L_1 \cap \bar{L}_2 | L_1 \in \mathsf{NP}, L_2 \in \mathsf{NP}\}$,
- $\mathsf{BH}_{2k+1} = \{L_1 \cup L_2 | L_1 \in \mathsf{BH}_{2k}, L_2 \in \mathsf{NP}\}$ with $k \geq 1$,
- $\mathsf{BH}_{2k+2} = \{L_1 \cap \bar{L}_2 | L_1 \in \mathsf{BH}_{2k+1}, L_2 \in \mathsf{NP}\}$, with $k \geq 1$,
- $\mathsf{coBH}_k = \{L | \bar{L} \in \mathsf{BH}_k\}$, with $k \geq 0$.

Thus, $\mathsf{BH}_1 = \mathsf{NP}$ and $\mathsf{coBH}_1 = \mathsf{coNP}$. The *boolean hierarchy* BH is the union of all BH_k (for k integer). It turns out to be equivalent to the class of problems that can be solved in deterministic polynomial time using a constant number of calls to an NP-oracle.

3 What's in Consequence Finding?

In this section, the four families of production fields that are considered extensively in the rest of this chapter are first presented; corresponding to each of these families of production fields, four notions of prime implicates, intended to represent the logically strongest characteristic theorems of the knowledge base, are pointed out. We successively give formal definitions for both notions, put the stress on their computation and consider various complexity issues related to them.

3.1 Introduction

As evoked previously, the purpose of consequence finding is to derive extensionally a set of characteristic theorems, those formulas that belong to the deductive closure of a knowledge base and to a given production field. To achieve this goal, a first problem that must be addressed is the representation one: how to represent such a set of theorems, which can be very huge (and even non finite)? In many cases, what is really important is the logical contents of the set of characteristic theorems, i.e., the syntactic form of the theorems is not central and every set of characteristic theorems sharing the same deductive closure can do the job. Thus, so as to minimize redundancy and achieve a concise representation of the set of characteristic theorems, focusing on the logically strongest ones is sufficient. Siegel [Siegel, 1987] defines the notion of *minimal precursor* of a set of formulas Φ as any subset Σ of Φ s.t. every formula of Φ is a logical consequence of a formula of Σ and no formula of Σ is a logical consequence of another formula of Σ. Clearly enough, every minimal precursor of Φ is logically equivalent to Φ. Unfortunately, every set of formulas does not have a minimal precursor. Nevertheless, this problem disappears whenever clauses are considered instead of (unconstrained) formulas: every set of propositional clauses has a minimal precursor [Siegel, 1987]. This is why a production field is typically considered as a set of clauses [Siegel, 1987] [Inoue and Helft, 1990]. This imposes that the theorems of interest are clauses as well but this is not really a restriction from a logical point of view since for every set of formulas, there exists an equivalent set of clauses [3]. Anyway, this is the restriction under which the main results about consequence finding that have been obtained so far hold, hence we shall adhere to it up to the end of the chapter. Thus, from now on, the notions of production field and of characteristic clause are defined as follows:

Definition 3.1 (production field, characteristic clause)

- *A* production field *is any subset* Π *of clauses from* $PROP_{PS}$*.*
- *Given a production field* Π *and a formula* Σ *from* $PROP_{PS}$*, a* characteristic clause γ *of* Σ *w.r.t.* Π *is a clause* γ *from* $PROP_{PS}$ *s.t.* $\Sigma \models \gamma$ *and* $\gamma \in \Pi$*.*

Example 3.1 *Let* Π *be the set of all clauses of size* 1 *from* $PROP_{PS}$ *that are not equivalent to a boolean constant and let* Σ *be the formula* $(a \Rightarrow b) \wedge (\neg a \Rightarrow b)$*. The set of characteristic clauses of* Σ *w.r.t.* Π *is* $\{b, true\}$*.*

In the following, we successively focus on four different families of production fields, from the simplest one to a more sophisticated one. The first

[3] This restriction may nevertheless have a great influence on the size of the representation of the set (hence, on some computational issues).

production field under consideration is the largest possible one, namely the set of all clauses from $PROP_{PS}$. Then, we shall successively turn to two important restrictions of it: the set of all clauses built up from a given set of literals and the set of all clauses that are not logical consequences of some formula. Finally, the combination of the two restrictions will be considered: the clauses of interest are those built up from a given set of literals and not implied by a given formula. This last family of production fields (parametrized by a set of literals and a formula) is the most interesting one for common sense reasoning (among those considered in this chapter [4]); moreover, it includes the other families as specific cases. However, all the four families listed in this chapter have their own utility. For instance, focusing on the clauses built up from a preset language is helpful in mechanizing diagnosis; especially, deriving what is known about the component states (e.g., the set of conflicts of a system, or the set of its positive conflicts) is required in many approaches to model-based diagnosis [Reiter, 1987]. Furthermore, consequence finding algorithms w.r.t. these four families of production fields are strongly dependent one another; in particular, computing standard prime implicates is a way to derive the more refined notions of prime implicates, and *vice-versa*. Considering all four families of production fields in an increasing order of complexity can thus prove a good strategy for envisioning the scope of the most sophisticated one.

A common factor to the four families of production fields under consideration is that each of them is stable (also called closed under deduction):

Definition 3.2 (stable production field) *A production field* Π *is* stable *(for logical deduction) iff every formula that is logically stronger than a formula from* Π *belongs to* Π *as well.*

Clearly enough, all production fields are not stable. Thus, the production field considered in the example above is not stable (e.g., $false$ does not belong to it).

Stability for logical deduction is a valuable property for consequence finding since it shows sufficient to focus on the logically strongest consequences of Σ: every logically strongest characteristic clause of Σ w.r.t. a stable production field Π is among the logically strongest clauses that are implied by Σ. More precisely, in order to generate a minimal precursor of the set of characteristic clauses of Σ w.r.t. a stable production field Π, it

[4] In this chapter, we shall not focus on the family of production fields consisting of all the clauses of bounded length, despite the interest of such a notion in designing polynomially-sized approximations of knowledge bases. The reader should look at [del Val, 1995] [del Val, 1996] [del Val, 1999] for details. Let us nevertheless note that some of the consequence finding algorithms described in this chapter - especially, SOL-resolution [Inoue, 1992] and SFK-resolution [del Val, 1999] - can be used to compute such characteristic theorems since they can take into account any (recursive) stable production field, and this is the case for the set of all clauses of bounded length.

it sufficient to *filter out selectively* the logically strongest clauses implied by Σ and belonging to Π (see e.g., [Castell, 1997]). In particular, the empty clause is (up to logical equivalence) the unique characteristic clause of any contradictory Σ w.r.t. any stable production field.

In the general case, it is not guaranteed that a minimal precursor of the set of characteristic clauses of Σ w.r.t. a stable production field Π is irredundant. It can easily be the case that a clause from this set is a logical consequence of all the remaining ones. Let us illustrate it on an example:

Example 3.2 *Let Π be the set of all clauses from $PROP_{PS}$ and Σ be the formula $(a \Rightarrow b) \wedge (b \Rightarrow c)$. The set of logically strongest characteristic clauses of Σ w.r.t. Π is $\{(\neg a \vee b), (\neg b \vee c), (\neg a \vee c)\}$ (up to logical equivalence). Clause $(\neg a \vee c)$ is a logical consequence of $\{(\neg a \vee b), (\neg b \vee c)\}$, hence it can be soundly removed from the set while preserving logical equivalence.*

Actually, exponentially more compact representations of the set of characteristic theorems of Σ w.r.t. Π can be achieved in the worst case (see section 3.4). However, such representations will not be considered in the following. There are several reasons for it. First, in some situations, especially when consequence finding is used for achieving some forms of abduction, the desired result is the whole set of logically strongest characteristic clauses (and any equivalent set does not do the job [5]). Additionally, from the whole set of logically strongest characteristic clauses of a knowledge base, we can check efficiently whether or not a given clause is a characteristic clause; such an efficiency cannot be guaranteed whenever any equivalent set is considered. Next, the filtering process described above for generating characteristic clauses would not necessarily work if any CNF representation of the set of logically strongest consequences of Σ were considered. Stepping back to the example above, assuming that the set of all logically strongest consequences of Σ has been simplified ($(\neg a \vee c)$ has been removed from the set), it is no longer possible to filter out a representation of the set of characteristic theorems of Σ w.r.t. the production field Π consisting of all clauses built up from literals $\neg a$ and c, only. Finally, every irredundant CNF representation of the set of characteristic clauses of Σ w.r.t. Π consists of logically strongest clauses of Σ. Computing first the whole set of such clauses is required by many simplification algorithms.

All this stuff explains the important role taken by the notion of *prime implicate* within consequence finding: the prime implicates of a formula Σ are just the main (prime) consequences (implicates) of it, i.e., the logically strongest clauses that are implied by Σ. Beyond consequence finding, the notion of prime implicate and/or the dual notion of prime implicant is

[5] Except in the context of probabilistic argumentation systems [Kohlas *et al.*, 1999a] where computing all the minimal abductive explanations of a formula is just a mean to derive its degree of support but is not mandatory, see [Kohlas *et al.*, 1999b].

a key notion of boolean algebra and propositional logic. Originally, during the 1950's, these concepts have been investigated in computer science for the purpose of circuit minimization (or simplification). Every computer scientist has heard of Karnaugh's maps [Karnaugh, 1953] and Quine–McCluskey's simplification algorithm [McCluskey, 1956] during his cursus. Nowadays, prime implicates/implicants are –under various names– at the core of numerous researches in many fields of computer science, where propositional logic is used as a representation language. Especially, they play an important role in fault tree analysis for reliability modelling of complex systems [Lee *et al.*, 1985] (in such a context, prime implicants are called minimal cut sets).

Several refinements of the basic notion of prime implicate/implicant have been pointed out so far in artificial intelligence. A notion of $\mathcal{L}_T$-prime implicate, where $\mathcal{L}_T$ is any subset of $PROP_{PS}$ (a production field), has been defined in [del Val, 1995]. A similar notion of selective minimal implicate has been introduced in [Kean, 1992], and instanciated into query-based minimal implicates, approximated query-based minimal implicates, ATMS implicates, length-based implicates. The concept of Π-prime implicate, i.e., the prime implicates from a stable production field Π, has been defined in [Castell, 1997]. A notion of prime $\mathcal{M}$-implicate has been pointed out in [Raiman and de Kleer, 1992]. A notion of theory prime implicate [Marquis, 1995] [Marquis and Sadaoui, 1996] has been introduced for the purpose of knowledge compilation, etc.

In the following, the stress is laid on three extensions of the standard notion of prime implicate that capture (respectively) the notions of logically strongest characteristic clause w.r.t. the three non-trivial families of production fields we consider: theorems from a prescribed set of literals L, theorems that appear only when some new information Σ is incorporated in the knowledge base Φ, and the combination of both; we call them respectively L–prime implicate, Φ–prime implicate and $\langle L, \Phi\rangle$–prime implicates. The focus is laid on the computation of such implicates. Because many algorithms have been proposed so far for computing prime implicants/implicates (let us mention among others the approaches by [Castell, 1996] [Castell, 1997] [de Kleer, 1992] [Dunham and Fridshal, 1959] [Fridshal, 1957] [Hwa, 1974] [Inoue, 1990] [Jackson, 1992] [Jackson and Pais, 1990] [Karnaugh, 1953] [Kean and Tsiknis, 1990b] [Madre and Coudert, 1991] [Madre and Coudert, 1994] [McCluskey, 1956] [Necula, 1967] [Ngair, 1993] [Palopoli *et al.*, 1999] [Pizzuti, 1996] [Quine, 1955] [Ramesh and Murray, 1993] [Ramesh *et al.*, 1997] [Rymon, 1994] [Socher, 1991] [Shiny and Pujari, 1998] [Shiny and Pujari, 1999] [Slagle *et al.*, 1970] [Strzemecki, 1992] [Tison, 1967]), it is not possible to describe them all hereafter. Since many of them are just variants of others, we focus instead on the metatheoretic properties on which they are based.

3.2 Definitions

Let us first define formally the four notions of prime implicate to be considered.

3.2.1 Prime implicates

The original definitions of the notions of prime implicates/implicants are due to W.V.O. Quine [Quine, 1952] [Quine, 1955] [Quine, 1959]. Quine was interested in simplifying truth functions, and he showed the prominent role of prime implicants/implicates in this objective. Accordingly, Quine's definitions concern truth functions. Alternatively, we present hereafter a definition of the concept of prime implicate of a propositional formula, considered up to logical equivalence.

Definition 3.3 (implicate, prime implicate) *Let Σ be a formula from $PROP_{PS}$.*

- *A clause π from $PROP_{PS}/\equiv$ is an* implicate *of Σ iff $\Sigma \models \pi$ holds.*
- *A clause π from $PROP_{PS}/\equiv$ is a* prime implicate *of Σ iff*
  - *π is an implicate of Σ, and*
  - *for every implicate π' of Σ, if $\pi' \models \pi$ holds, then $\pi \models \pi'$ holds.*

As the previous definition states it formally, the implicates of Σ are the clauses that are consequences of Σ and the prime implicates of Σ are its minimal implicates w.r.t. the partial ordering $\models$ over $PROP_{PS}/\equiv$, i.e., the logically strongest ones. The definition can be extended in a straightforward way to the first-order case by considering first-order clauses, i.e., universally quantified finite disjunctions of literals.

In the following, the set of prime implicates of a propositional formula Σ is denoted $PI(\Sigma)$. Note that every clause from $PROP_{PS}/\equiv$ is an equivalence class, the set of all clauses from $PROP_{PS}$ that are logically equivalent. For the sake of simplicity, each equivalence class is given by one of its representatives, i.e., a clause from $PROP_{PS}$, and such clauses are also called the prime implicates of Σ.

Example 3.3 *Let $\Sigma = (a \Rightarrow b) \wedge (b \Rightarrow c)$. $PI(\Sigma) = \{(\neg a \vee b), (\neg b \vee c), (\neg a \vee c)\}$ (up to logical equivalence).*

Clearly enough, $PI(\Sigma)$ is a minimal precursor of the set of characteristic clauses of Σ w.r.t. the stable production field Π consisting of all clauses from $PROP_{PS}$.

Two extreme situations deserve some additional comments. In the case where Σ is valid, only valid clauses (i.e., tautologies) are implicates of Σ.

Subsequently, $PI(\Sigma)$ contains only the class of such clauses, one representative of which being the boolean constant *true*. In the case where Σ is unsatisfiable, every clause is an implicate of Σ. In this situation, $PI(\Sigma)$ contains only one class, represented by the empty clause, or equivalently by the boolean constant $false$.

While prime implicate can be considered a standard notion at a first glance, this is actually far from being true. Indeed, many non-equivalent definitions of prime implicates can be found in the literature. Thus, in some papers, including Quine's seminal ones [Quine, 1952] [Quine, 1955] [Quine, 1959], the notion of prime implicate is given a more syntactic view; especially, logical entailment between clauses is weakened into set-theoretic inclusion (and sometimes clauses are considered as sets of literals). Though it is close to Definition 3.3, such a characterization formally differs from it in several ways. Especially, syntax-independence is no longer satisfied: two different tautologies are in many situations not comparable w.r.t. set-theoretic inclusion, hence they may have different sets of prime implicates. In the same vein, boolean constants are not handled in a satisfying way within such a definition. In some other papers (e.g., [Reiter and de Kleer, 1987]), the prime implicates of Σ are defined as the implicates of Σ that are minimal w.r.t. $\subseteq$; thus, $b \vee \neg b$ is viewed as a prime implicate of a. Elsewhere (e.g., [Tsiknis and Kean, 1988]), such clauses are called the minimal implicates of Σ while its prime implicates are defined as those implicates π of Σ s.t. no implicate π' of Σ except π is s.t. $\pi' \models \pi$. Within such a definition (that clearly has a syntactic flavor even if clauses are viewed as sets of literals), valid formulas do not have prime implicates.

While the notion of prime implicate is central for consequence finding, the dual notion, namely *prime implicant*, is sometimes considered more convenient. Intuitively, a prime implicant of a propositional formula is one of the weakest states of truth in which the formula is true. Formally:

Definition 3.4 (implicant, prime implicant) *Let Σ be a formula from $PROP_{PS}$.*

- *A term π from $PROP_{PS}/\equiv$ is an* implicant *of Σ iff $\pi \models \Sigma$ holds.*
- *A term π from $PROP_{PS}/\equiv$ is a* prime implicant *of Σ iff*
  - *π is an implicant of Σ, and*
  - *for every implicant π' of Σ, if $\pi \models \pi'$ holds, then $\pi' \models \pi$ holds.*

Duality has here a strong, formal meaning, made precise by the next proposition:

Proposition 3.1 *Let Σ be a formula from $PROP_{PS}$. Every prime implicate of Σ is the negation of a prime implicant of $\neg\Sigma$ and every prime implicant of Σ is the negation of a prime implicate of $\neg\Sigma$.*

Example 3.4 *Let* $\Sigma = (a \Leftrightarrow b)$. $PI(\Sigma) = \{(\neg a \vee b), (\neg b \vee a)\}$. *Terms* $(a \wedge \neg b)$ *and* $(\neg a \wedge b)$ *are the negations of the prime implicates of* Σ *and the prime implicants of* $\neg\Sigma$ *(up to logical equivalence).*

Thus, the prime implicates (resp. implicants) of a formula can be easily recovered from the prime implicants (resp. implicates) of its negation (negating them is sufficient). Since the mapping between both sets is bijective, the two sets are similar, i.e., the number of prime implicates of Σ is the number of prime implicants of $\neg\Sigma$. More importantly, all the properties about prime implicates that are listed in the following admit some dual versions related to prime implicants. Additionally, every algorithm for computing prime implicates can be easily adapted to derive prime implicants. Since we are specifically interested in consequence finding, both properties and algorithms will be specified in terms of prime implicates (even if their original presentation was not).

3.2.2 L–prime implicates

The prime implicates of a propositional formula characterize *all its implicates*, i.e., all the clauses that can be soundly derived from it. Let us now introduce a more refined notion of (prime) implicate that proves useful for our purpose of characterizing some consequences, only. It is related to the case where the clauses built up from a prescribed set of literals are of interest.

Definition 3.5 (L–implicate, L–prime implicate) *Let* L *be a finite subset of* L_{PS} *and* Σ *be a formula from* $PROP_{PS}$.

- *A clause* π *from* $PROP_{PS}/\equiv$ *is a* L–implicate *of* Σ *iff* $\Sigma \models \pi$ *holds and* $Lit(\pi) \subseteq L$.
- *A clause* π *from* $PROP_{PS}/\equiv$ *is a* L–prime implicate *of* Σ *iff*
  - π *is a* L*–implicate of* Σ, *and*
  - *for every* L*–implicate* π' *of* Σ, *if* $\pi' \models \pi$ *holds, then* $\pi \models \pi'$ *holds.*

In the following, $PI_L(\Sigma)$ denotes the set of L–prime implicates of Σ. This set includes the logically strongest L–implicates of Σ, i.e., the logically strongest implicates of Σ that are built up from L.

Example 3.5 *Let* $L = \{\neg a, b\}$ *and* $\Sigma = (a \Rightarrow b) \wedge (b \Rightarrow c)$. $PI_L(\Sigma) = \{(\neg a \vee b)\}$. *Contrastingly,* $PI_{\{a,c\}}(\Sigma) = \emptyset$.

$PI_L(\Sigma)$ is a minimal precursor of the set of characteristic clauses of Σ w.r.t. the production field Π corresponding to the set of all clauses γ from $PROP_{PS}$ s.t. $Lit(\gamma) \subseteq L$.

In many situations, the whole set of prime implicates of a formula that are built up from a given set V of variables is of interest, i.e., the literals signs do not matter. Nevertheless, we have chosen putting the stress on the notion of L–prime implicate instead. On the one hand, it is more general than the former notion of V–prime implicates (the V–prime implicates of Σ are its L_V–prime implicates). On the other hand, the literal signs have to be taken into account in various cases (e.g., in the consistency-based diagnosis framework, the positive conflicts of a system are to be considered, only; when dealing with abduction, it can be the case that the set of possible individual hypotheses is not closed under negation; when implementing circumscription, the literals signs have a great importance, etc.).

3.2.3 Φ–prime implicate

Another refinement of basic prime implicates is the notion of Φ–prime implicate. It proves particularly helpful in the situation where the available knowledge is not given as a whole, but in an incremental way, by keeping separate some background knowledge represented as a propositional formula Φ, and some incoming knowledge represented as a formula Σ. It aims at characterizing the *new consequences* of Φ when expanded with Σ, i.e., the clauses that are consequences of $(\Phi \wedge \Sigma)$ but are not consequences of Φ. Such clauses are also called new clauses in [Inoue and Helft, 1990] [Inoue, 1992][6] and direct consequences in [Kean and Tsiknis, 1990a] [Kean, 1992] [Kean and Tsiknis, 1993].

Definition 3.6 (Φ–implicate, Φ–prime implicate) *Let Φ and Σ be two formulas from $PROP_{PS}$.*

- *A clause π from $PROP_{PS}/\equiv$ is a* Φ–implicate *of Σ iff $\Phi \wedge \Sigma \models \pi$ holds and $\Phi \not\models \pi$ holds.*
- *A clause π from $PROP_{PS}/\equiv$ is a* Φ–prime implicate *of Σ iff*
  - *π is a Φ–implicate of Σ, and*
  - *for every Φ–implicate π' of Σ, if $\pi' \models \pi$ holds, then $\pi \models \pi'$ holds.*

$PI^{\Phi}(\Sigma)$ denotes the set of Φ–prime implicates of Σ. Such clauses are the logically strongest Φ–implicates of Σ.

Example 3.6 *Let $\Phi = (a \Rightarrow b)$ and $\Sigma = (b \Rightarrow c)$. $PI^{\Phi}(\Sigma) = \{(\neg b \vee c), (\neg a \vee c)\}$ (up to logical equivalence). Thus, while $(\neg a \vee b)$ is a logical consequence of $(\Phi \wedge \Sigma)$, it cannot be considered as a new one since it already appears in the deductive closure of Φ.*

[6] Often, these formulas are not considered as characteristic clauses, i.e., the set of clauses that are not logical consequences of Φ is not specifically viewed as a production field, but as new characteristic clauses.

$PI^{\Phi}(\Sigma)$ is a minimal precursor of the set of characteristic clauses of $\Phi \wedge \Sigma$ w.r.t. to the production field Π consisting of every clause from $PROP_{PS}$ that is not a logical consequence of Φ.

As a straightforward consequence of Definition 3.6, $false$ is (up to logical equivalence) the unique Φ–prime implicate of Σ whenever Φ is satisfiable but $(\Phi \wedge \Sigma)$ is not.

3.2.4 $\langle L, \Phi \rangle$–prime implicate

Clearly enough, the notions of L–implicate and Φ–implicate can be easily combined to give birth to the notion of $\langle L, \Phi \rangle$–prime implicate. Indeed, every Φ–prime implicate of Σ that is equivalent to a clause built up from L is a $\langle L, \Phi \rangle$–prime implicate of Σ, and every $\langle L, \Phi \rangle$–prime implicate of Σ can be characterized in this way. Formally:

Definition 3.7 ($\langle L, \Phi \rangle$–implicate, $\langle L, \Phi \rangle$–prime implicate) *Let L be a finite subset of L_{PS} and Φ and Σ be formulas from $PROP_{PS}$.*

- *A clause π from $PROP_{PS}/\equiv$ is a $\langle L, \Phi \rangle$–*implicate *of Σ iff π is a Φ–implicate of Σ and $Lit(\pi) \subseteq L$.*
- *A clause π from $PROP_{PS}/\equiv$ is a $\langle L, \Phi \rangle$–*prime implicate *of Σ iff*
  - *π is a $\langle L, \Phi \rangle$–implicate of Σ, and*
  - *for every $\langle L, \Phi \rangle$–implicate π' of Σ, if $\pi' \models \pi$ holds, then $\pi \models \pi'$ holds.*

$PI_L^{\Phi}(\Sigma)$ denotes the set of $\langle L, \Phi \rangle$–prime implicates of Σ. $\langle L, \Phi \rangle$–prime implicates are called *new characteristic clauses* in [Inoue and Helft, 1990] [Inoue, 1992].

Example 3.7 *Let $L = \{\neg a, c\}$, $\Phi = (a \Rightarrow b)$ and $\Sigma = (b \Rightarrow c)$. $PI_L^{\Phi}(\Sigma) = \{(\neg a \vee c)\}$ (up to logical equivalence).*

$PI_L^{\Phi}(\Sigma)$ is a minimal precursor of the set of characteristic clauses of $(\Phi \wedge \Sigma)$ w.r.t. the production field Π consisting of every clause γ of $PROP_{PS}$ s.t. $Lit(\gamma) \subseteq L$ and γ is not a logical consequence of Φ.

As already evoked, the notion of $\langle L, \Phi \rangle$–prime implicate is central for several forms of common sense reasoning, including abduction, simple default reasoning, circumscription and related forms of closed world reasoning. We shall return to these issues later.

3.3 Computing prime implicates

In this section, some metatheoretic properties of the notions of prime implicates presented above are given. These properties make precise the scope of both notions and some of them are the corner stones on which the great majority of the algorithms for computing prime implicates that have been proposed so far are based.

3.3.1 Prime implicates

One of the most salient feature of the notion of prime implicate is that it is independent from the syntax. Indeed, the syntactic presentation of Σ has no influence on its prime implicates.

Proposition 3.2 *Let Σ and Ψ be two formulas from $PROP_{PS}$. $\Sigma \equiv \Psi$ holds iff $PI(\Sigma) = PI(\Psi)$.*

Thus, two equivalent propositional formulas *have the same prime implicates*. This is why the set of prime implicates of Σ is sometimes called the *prime implicates normal form* or *Blake's normal form* of Σ [Blake, 1937]. Moreover, this shows our definition to cohere with Quine's one (since every boolean function can be viewed as an equivalence class of propositional formulas w.r.t. logical equivalence). Since only a finite subset of variables from PS may occur in Σ, we have as a corollary:

Corollary 3.1 *$PI(\Sigma)$ always is a finite set.*

Note that this corollary would not be satisfied for valid formulas Σ, had the set inclusion-based definition of prime implicate been considered. Furthermore, it is not satisfied in first-order logic (even in the case of a first-order language without equality and without functional symbols, except constants) [Marquis, 1993].

Since the semantics of any propositional formula only depends on the variables occurring in it, the set of symbols over which $PI(\Sigma)$ is generated can be restricted to $Var(\Sigma)$: every prime implicate of Σ has a representative in $PROP_{Var(\Sigma)}$. Moreover, every literal of a prime implicate of Σ belongs to $Lit(\Sigma)$, the set of literals occurring in the NNF of Σ. An even more accurate characterization of the language of $PI(\Sigma)$ can be achieved by focusing on the subset $Dep(\Sigma)$ of $Lit(\Sigma)$ containing the literals Σ depends on, i.e., those appearing in every syntactic presentation of Σ (see [Lang *et al.*, 1999] for details). Formally:

Definition 3.8 (literal independence) *Let Σ be a formula from $PROP_{PS}$ and $x \in PS$.*

- Σ *is* independent from x *iff* $\Sigma \models \Sigma_{x \leftarrow false}$ *holds,*

- Σ *is* independent from $\neg x$ *iff* $\Sigma \models \Sigma_{x \leftarrow true}$ *holds.*

If Σ is not independent from literal l, it is said *dependent on* it. Clearly enough, Σ can easily depend on both l and $\neg l$, but this is not mandatory. $Dep(\Sigma)$ denotes the subset of $Lit(\Sigma)$ containing the literals l Σ is dependent on l.

Example 3.8 *Let* $\Sigma = (\neg a \vee b) \wedge (a \vee c) \wedge (b \vee c \vee d)$. Σ *is independent from* $\neg b$, $\neg c$, d, $\neg d$. Σ *is dependent on* $\neg a$, b, a, *and* c.

This notion of literal independence slightly generalizes the notion of variable independence presented in [Lang and Marquis, 1998]. Indeed, Σ is independent from variable x iff Σ is independent from literal x and Σ is independent from literal $\neg x$. So as to compute the prime implicates of Σ, it is sufficient to focus on the literals on which Σ is dependent.

Proposition 3.3 *Let* Σ *be a formula from* $PROP_{PS}$. *Every prime implicate of* Σ *is equivalent to a clause containing literals from* $Dep(\Sigma)$, *only.*

Example 3.9 *Let* $\Sigma = (\neg a \vee b) \wedge (a \vee c) \wedge (b \vee c \vee d)$. *We have* $Dep(\Sigma) = \{\neg a, b, a, c\}$ *and* $PI(\Sigma) = \{(\neg a \vee b), (a \vee c), (b \vee c)\}$ *(up to logical equivalence).*

Proposition 3.2 is valuable for the purpose of consequence finding since equivalent formulas obviously share the same consequences. The next *covering property* also plays a key role w.r.t. this issue: it shows that, in some sense, every consequence of Σ is captured by its prime implicates.

Proposition 3.4 (covering) *Let* Σ *be a formula from* $PROP_{PS}$ *and* γ *be a clause from* $PROP_{PS}$. $\Sigma \models \gamma$ *holds iff* $(\exists \pi \in PI(\Sigma))(\pi \models \gamma)$.

This proposition would not hold, had the set inclusion-based definition of prime implicate been considered. It is also not satisfied in the first-order case [Marquis, 1993], but can be achieved by modifying the definition of first-order prime implicates, especially by considering that the prime implicates of a first-order formula are its minimal implicates w.r.t. θ-subsumption (a decidable restriction of $\models$) [Inoue, 1992].

Thanks to Proposition 3.4, checking whether a formula Ψ in CNF is a logical consequence of Σ comes down to checking whether every clause γ of Ψ is entailed by a prime implicate π of Σ. Such tests can be easily performed in time polynomial in the size of $PI(\Sigma)$ plus the size of γ, since $\pi \models \gamma$ holds iff γ is valid or π is contradictory or every literal of π (except $false$ that can be soundly removed from it) is a literal of γ. As a straightforward consequence of the covering property, no information is lost by replacing Σ by its prime implicates. Accordingly, $PI(\Sigma)$ can be considered as an *equivalence-preserving compilation* of Σ [Reiter and de Kleer, 1987] [Cadoli and Donini, 1998].

Corollary 3.2 $\Sigma \equiv PI(\Sigma)$.

As evoked previously, it is not the case that every CNF formula equivalent to Σ satifies the covering property. In particular, no prime implicate cover of Σ (i.e., any subset of $PI(\Sigma)$ that is equivalent to Σ) satisfies this property, unless it consists of the set of prime implicates of Σ itself.

Example 3.10 *Let* $\Sigma = (a \Rightarrow b) \wedge (b \Rightarrow c)$. *The set of clauses* $\{(\neg a \vee b), (\neg b \vee c)\}$ *is an irredundant prime implicate cover of* Σ. *The clause* $(\neg a \vee c)$ *(which is a prime implicate of* Σ*) is not entailed by any clause of this cover.*

A direct consequence of the covering property is a relationship between the prime implicates of a formula Σ and those of any logically weakest formula Ψ. Indeed, if $\Sigma \models \Psi$ holds, then for every prime implicate π_Ψ of Ψ, there exists a prime implicate π_Σ of Σ s.t. $\pi_\Sigma \models \pi_\Psi$ holds. In a formal way, $\models$ is a surjective relation over $PI(\Sigma) \times PI(\Psi)$ iff $\Sigma \models \Psi$. However, logical entailment between $PI(\Sigma)$ and $PI(\Psi)$ does not exhibit any other remarkable property in the general case. Especially, it is not functional and it is not injective. Hence, strengthening (or weakening) a formula may dramatically change the set of its prime implicates (and the number of its elements, especially in the case it makes the formula contradictory or valid).

Example 3.11

- *Let* $\Sigma = a$ *and* $\Psi = (a \vee b) \wedge (a \vee c)$. *Then the prime implicate* a *of* Σ *entails both prime implicates* $(a \vee b)$ *and* $(a \vee c)$ *of* Ψ.
- *Let* $\Sigma = (a \wedge b)$ *and* $\Psi = (a \vee b)$. *Both prime implicates* a *and* b *of* Σ *entail the prime implicate* $(a \vee b)$ *of* Ψ.

Let us now focus on the computational aspects of prime implicates. There are many approaches for computing prime implicates or prime implicants. From an historical perspective, the first methods that have been pointed out are based on propositional resolution (although they have been put forward much before Robinson's resolution principle [Robinson, 1965]). Quine [Quine, 1959] introduces a notion of *iterated consensus* between clauses so as to derive prime implicates [7]. By repeteadly computing consensus from the clauses of a CNF formula Σ and simplifying the set of clauses by keeping only the logically strongest ones, the prime implicates of Σ can be derived.

Let us now formally define *resolution* as the inference rule relating two clauses with their resolvent. For the sake of simplicity [8], reduced clauses (cf. Section 2.1) will be considered, only:

[7] Actually, Quine was interested in deriving prime implicants; hence, consensus, which can be traced back to the work of Blake [Blake, 1937], is dual of resolution.

[8] Dealing with reduced clauses enables considering a single inference rule between clauses, namely resolution. Otherwise, as in the general case of clausal first-order logic, it is

Definition 3.9 (resolution) *Let γ_1 and γ_2 be two reduced clauses from $PROP_{PS}$.*

- *γ_1 and γ_2 can be* resolved upon variable *x from PS iff $x \in Lit(\gamma_1)$ and $\neg x \in Lit(\gamma_2)$.*
- *if γ_1 and γ_2 can be resolved upon x, their* resolvent *is the clause obtained by reducing ${\gamma_1}_{x \leftarrow false} \vee {\gamma_2}_{x \leftarrow true}$.*

Example 3.12 *Let γ_1 be the clause $(a \vee \neg b)$, γ_2 be the clause $(b \vee c)$, and γ_3 be the clause $(\neg a \vee b)$. γ_2 and γ_1 can be resolved upon b. The corresponding resolvent is the clause $(c \vee a)$. Contrastingly, γ_2 and γ_3 cannot be resolved together. Finally, γ_1 and γ_3 can be resolved upon a, while γ_3 and γ_1 can be resolved upon b. In these two cases, the corresponding resolvent is $true$.*

Note that the resolvent of two clauses γ_1 and γ_2 over variable x is equivalent to what could be the resolvent of γ_2 and γ_1 over $\neg x$. In other words, the ordering in which two clauses are resolved upon does not really matter: what is important is the literal upon which they are resolved. Accordingly, in the following, two clauses γ_1 and γ_2 are said to be resolved upon literal l iff γ_1 and γ_2 are resolved upon l or γ_2 and γ_1 are resolved upon l (depending on the fact that l occurs positively or not in γ_1).

Clearly enough, the resolvent of two clauses is a logical consequence of their conjunction. Hence, expanding a set of clauses with resolvents of clauses from it is a truth-preserving operation: resolvents are theorems.

It is quite easy to see that, whenever two clauses can be resolved upon more than one variable, the corresponding resolvent is a tautology. Accordingly, the resolvent of two clauses is unique up to the ordering of the literals occurring in it.

Definition 3.10 (resolution proof) *Let Σ be a formula from $PROP_{PS}$ in CNF s.t. every clause occurring in it is reduced. Let γ be a clause from $PROP_{PS}$. A* resolution proof *of γ from Σ is a finite sequence γ_1, ..., γ_n of clauses from $PROP_{PS}$ s.t. for every $i \in 1 .. n$, γ_i is a clause from Σ or γ_i is the resolvent of γ_j and γ_k with $j < i$ and $k < i$, and $\gamma_n = \gamma$.*

For every $i, j \in 1 .. n$, when $i \leq j$, γ_i is said an ancestor *of γ_j and γ_j is said a* descendant *of γ_i.*

We note $\Sigma \vdash \gamma$ whenever γ has a resolution proof from Σ.

Example 3.13 *Let γ_1 be the clause $(a \vee \neg b)$, γ_2 be the clause $(b \vee c)$, and γ_3 be the clause $(\neg c \vee d)$. We have $\gamma_1 \wedge \gamma_2 \wedge \gamma_3 \vdash (a \vee d)$. Here is a corresponding resolution proof:*

necessary to introduce another inference rule between clauses, called *factorisation*. This corresponds to a more syntactic view of resolution, but the price to be paid is a more complex presentation which is unnecessary in the propositional case.

1. γ_1 *(input clause);*

2. γ_2 *(input clause);*

3. $(a \vee c)$ *(resolvent of* γ_1 *and* γ_2*);*

4. γ_3 *(input clause);*

5. $(a \vee d)$ *(resolvent of* γ_3 *and* $(a \vee c)$*).*

Typically, a resolution strategy is defined by adding some deletion rules to resolution (e.g., the tautology deletion rule, the subsumed clauses deletion rule, the pure literal deletion rule) together with an application policy for rules that limits their application, hence the set of admissible proofs w.r.t. the strategy. For our purpose, what is important is the set of theorems that can be derived according to the strategy:

Definition 3.11 (resolution strategy) *A* resolution strategy *is a mapping R from the set of all CNF formulas Σ from $PROP_{PS}$ consisting of reduced clauses into itself s.t. $R(\Sigma) \subseteq \{\gamma; \Sigma \vdash \gamma\}$. We note $\Sigma \vdash_R \gamma$ whenever $\gamma \in R(\Sigma)$.*

The *general resolution* strategy G is defined by $G(\Sigma) = \{\gamma; \Sigma \vdash \gamma\}$. Clearly enough, every resolution strategy R is a restriction of the general resolution strategy in the sense that $R(\Sigma) \subseteq G(\Sigma)$ for every Σ. It is worth noting that, in contrast to the first-order case, $G(\Sigma)$ can only contain a finite number of (reduced) clauses, and this is *a fortiori* the case for every resolution strategy R. However, the number of non-equivalent clauses in $G(\Sigma)$ can be huge! This is why resolution strategies are helpful: efficient resolution strategies are those for which the search space is as small as possible, while containing all the theorems we are interested in.

Many resolution strategies have been proposed so far in the literature (see [Chang and Lee, 1973] [Loveland, 1978] for a survey). Consensus is one of them. By definition, the *consensus* of two clauses γ_1 and γ_2 that can be resolved upon variable x is their resolvent unless this one is a tautology. In this case, the consensus of γ_1 and γ_2 is not defined. A consensus proof is a resolution proof in which the consensus rule (that is a restriction of resolution) is applied at each step. Accordingly, $C(\Sigma)$ is the set of all clauses that have a consensus proof from Σ. Clearly enough, consensus corresponds to the (very simple) resolution strategy with tautology elimination. Interestingly, resolution and consensus can be used to generate prime implicates.

Proposition 3.5 (saturation) *Let Σ be a CNF formula consisting of reduced clauses. We have $PI(\Sigma) = min(G(\Sigma), \models) = min(C(\Sigma), \models)$ (up to logical equivalence).*

Example 3.14 *Let* $\Sigma = (a \vee \neg b) \wedge (b \vee c) \wedge (\neg c \vee d)$. *We have* $PI(\Sigma) = min(\{(a \vee \neg b), (b \vee c), (\neg c \vee d), (a \vee c), (b \vee d), (a \vee d)\}, \models) = \{(a \vee \neg b), (b \vee c), (\neg c \vee d), (a \vee c), (b \vee d), (a \vee d)\}$ *(up to logical equivalence).*

This proposition is the theoretical basis of Quine's iterated consensus algorithm [Quine, 1955]. In Quine's algorithm, the consensus rule is applied repeteadly: at each step, the consensus of two clauses in the set of current clauses (initialized as Σ) is added to this set. In contrast to what Proposition 3.5 could suggest [9], minimization is performed at each step: every clause that is not minimal w.r.t. $\models$ (i.e., it does not belong to the set of the logically strongest clauses generated so far) is removed. Quine's algorithm terminates when the current set of clauses is *saturated* for the successive applications of consensus and minimization, i.e., a fixed point is obtained: the set of clauses obtained after the successive applications of consensus and minimization is identical (up to the ordering of literals inside clauses) to the previous set of clauses. When the algorithm stops, the resulting set is $PI(\Sigma)$ (up to logical equivalence). In particular, every resolvent of two prime implicates of Σ is a logical consequence of a prime implicate of Σ (this is just a consequence of the covering property). Clearly, Quine's iterated consensus algorithm mainly corresponds to resolution with a systematic application of deletion of subsumed clauses and tautologies once all the resolvents at each step have been generated.

Many resolution-based algorithms can be designed for computing prime implicates. With this respect, an important issue is the choice of the resolution strategy. Indeed, can any resolution strategy be used so as to generate prime implicates using a saturation-based approach? Actually, the answer is no. In order to understand why, it is essential to make a distinction between two completeness properties that a resolution strategy can exhibit: *completeness in proof finding* (also called refutational completeness) and *completeness in consequence finding* (also called deductive completeness):

Definition 3.12 (proof finding vs. consequence finding) *Let* R *be a resolution strategy.*

- R *is* complete in proof finding *iff for every CNF formula* Σ *from* $PROP_{PS}$ *consisting of reduced clauses and every reduced clause* γ *from* $PROP_{PS}$, *we have* $\Sigma \models \gamma$ *iff* $\Sigma \wedge \neg\gamma \vdash_R \Box$.

- R *is* complete in consequence finding *iff for every CNF formula* Σ *from* $PROP_{PS}$ *consisting of reduced clauses and every reduced clause* γ *from* $PROP_{PS}$, *we have* $\Sigma \models \gamma$ *iff there exists a clause* Ψ *s.t.* $\Sigma \vdash_R \Psi$ *and* $\Psi \models \gamma$ *holds.*

[9] From a logical point of view, minimization is not mandatory at each step; however, from a computational point of view, this has a strong impact on the size of the instances that can be dealt with in practice with Quine's algorithm.

Typically (and historically), resolution is oriented towards proof finding: the aim is to be able to show that any finite set of clauses is contradictory (when it is the case) by deriving a *refutation* from it (i.e., a resolution proof of the empty clause). In other words, there is only one theorem of interest, the empty clause. Accordingly, for the purpose of proof finding, resolution strategies for which the sets of admissible proofs (hence the set of corresponding theorems) are as small as possible are the best ones (as long as they remain complete in proof finding). In order to achieve this goal, many improvements over general resolution have been achieved (see [Chang and Lee, 1973] [Loveland, 1978] for details). Among them are *ordered resolution strategies*; roughly, within such strategies, two clauses are to be resolved upon literal l only if either l or $\neg l$ is maximal (w.r.t. a given total literal ordering) in one of the two clauses. An A-ordering for a CNF formula Σ is a strict, total ordering over $Var(\Sigma)$ [10]. In ordered resolution with A-ordering, the maximal literal in any clause is maximal w.r.t. the A-ordering. Thus, all the clauses are considered ordered w.r.t. the same ordering. Contrastingly, in ordered resolution with C-ordering, every clause is considered as a sequence of literals, and the last literal in the sequence is the maximal literal of the clause.

Contrariwise to resolution strategies complete in proof finding, every clause that is a logical consequence of a CNF formula is accessible using a resolution strategy complete in consequence finding, in the sense that a logically stronger clause has a resolution proof.

Thus, for the purpose of computing prime implicates, the important notion is completeness in consequence finding. Indeed:

Proposition 3.6 *Let R be a resolution strategy that is complete in consequence finding. Let Σ be a CNF formula from $PROP_{PS}$ consisting of reduced clauses. For every $\pi \in PI(\Sigma)$, we have $\Sigma \vdash_R \pi$.*

This proposition (that also holds in the first order-case [Marquis, 1993]) shows that Quine's saturation-based approach to computing prime implicates (Proposition 3.5) can be extended to the case where any resolution strategy that is complete in consequence finding is considered. So it is important to identify such strategies.

The first step in this direction has been done by Lee [Lee, 1967]. Lee shows that general resolution G is complete in consequence finding (actually, Lee's subsumption theorem holds in the general case of clausal first-order logic: each time a non-valid clause γ is a logical consequence of a set of first-order clauses Σ, there is a resolution proof of a clause Ψ that subsumes γ).

[10] Such an ordering $x_1 < \ldots < x_n$ can be extended to the following total ordering over $Lit(\Sigma)$: $\neg x_1 < x_1 < \neg x_2 < x_2 < \ldots < \neg x_n < x_n$. This extension is to be considered only when a tautology is to be resolved which is not the case for many resolution strategies.

Clearly enough, every resolution strategy that is complete in consequence finding also is complete in proof finding. Indeed, if there exists a clause Ψ s.t. $\Sigma \vdash_R \Psi$ and $\Psi \models \gamma$ holds, every literal occurring in Ψ also occurs in γ (except if γ is a tautology); anyway, expanding Σ with the negations of literals of γ, it is possible to extend the resolution proof of Ψ into a resolution proof of the empty clause (refutation) from $\Sigma \wedge \neg\gamma$. However, the converse does not hold; there are resolution strategies that are complete in proof finding but incomplete in consequence finding. As an example, let us consider the *pure literal deletion rule* [Davis and Putnam, 1960]: assume that a given literal l occurs only positively (resp. negatively) in a CNF formula Σ, then every clause containing l can be removed from Σ since in the case where the resulting formula has a model, it can be extended into a model of Σ by assigning to l the truth value 1 (resp. 0). This inference rule preserves satisfiability, but not equivalence. Thus, modifying any resolution strategy complete in consequence finding by applying the pure literal deletion rule in priority (before any other rule of the strategy) results in a resolution strategy that is still complete in proof finding but not any longer in consequence finding.

Example 3.15 *Let $\Sigma = (a \vee \neg b) \wedge (b \vee c) \wedge (\neg c \vee d)$. Both a and d are pure within Σ, hence both clauses $(a \vee \neg b)$ and $(\neg c \vee d)$ can be removed from Σ without questioning its satisfiability. Removing such clauses in priority prevents from generating the prime implicates $(a \vee c)$, $(b \vee d)$, $(a \vee d)$ of Σ.*

Slagle, Chang and Lee [Slagle *et al.*, 1969] extend Lee's result to semantic resolution strategies [Slagle, 1967]. Especially, they show that I-semantic resolution, including positive and negative hyperresolution, and AI-resolution (i.e. ordered I-resolution with an A-ordering) are complete in consequence finding in some restricted senses. Especially, every prime implicate of a CNF formula Σ that is false in an interpretation I has a I-resolution proof from Σ (but a I-resolution proof is not guaranteed in general for the remaining prime implicates). A restricted form of completeness in consequence finding can also be achieved for set-of-support resolution [Wos *et al.*, 1967] (usually considered as another restriction of semantic resolution). In any set-of-support resolution proof, two clauses can be resolved upon only if at least one of them has an ancestor clause in the set of support (a subset T of the initial set of clauses Σ s.t. $\Sigma \setminus T$ is satisfiable). Clearly enough, completeness in consequence finding is not guaranteed for clauses that are logical consequences of Σ minus the set of support.

Minicozzi and Reiter [Minicozzi and Reiter, 1972] consider some linear resolution strategies. They show that some forms of linear resolution are complete in consequence finding while some others are not. To be more precise, linear resolution with merging, subsumption and tautology conditions, and ordered linear resolution with merging, A-ordering and tautology conditions are complete in consequence finding (provided that every possible A-ordering is considered, while completeness in proof finding is achieved for any A-ordering). Contrastingly, ordered linear resolution involving C-ordering is not complete in consequence finding in the general case. All the theorems of completeness in consequence finding reported in [Slagle *et al.*, 1969] and [Minicozzi and Reiter, 1972] hold in first-order clausal logic.

Based on resolution strategies that are complete in consequence finding, several refinements of Quine's brute-force algorithm have been pointed out so far in the literature [Tison, 1967] [Kean and Tsiknis, 1990b] [Jackson, 1992] [de Kleer, 1992]. A first improvement consists in avoiding generating several times the same clause (through different resolution proofs) by observing that the consensus operation is associative. Thus, in the example above, the clause $(a \vee d)$ can be derived by resolving first the first two input clauses, then their resolvent $(a \vee c)$ with the third input clause. It can also be derived by resolving first the clauses $(b \vee c)$ and $(\neg c \vee d)$, then resolving their resolvent $(b \vee d)$ with the first input clause. Tison [Tison, 1967] shows how such a redundancy can be avoided by generating consensi in an ordered way: given an A-ordering $x_1 < \ldots < x_n$, at step $i \in 1 \ldots n$, all the consensi of the current set of clauses over variable x_i are derived. Thus, consensi over the first variable x_1 are computed during the first step, then consensi over the second variable x_2 during the second step, and so on. The point is that when all the consensi over a variable from the current set of clauses have been computed at a given step, it will never be necessary to perform any consensus inference over that variable. Considering any A-ordering does not question the correctness of the algorithm in generating prime implicates but it may strongly bear on its efficiency.

As recently pointed out by del Val [del Val, 1999], Tison's algorithm can be viewed as implicitly exploring a specific set of resolution proofs, called *kernel deductions*. When considering kernel resolution proofs, every reduced clause γ is considered as a *s-clause*, i.e. an ordered pair $\langle skip(\gamma), kernel(\gamma)\rangle$ of disjoint sets of literals. From a logical point of view, γ is equivalent to the disjunction of all the literals belonging to $skip(\gamma) \cup kernel(\gamma)$. The notion of s-clause is just a way to mark some literals of a clause so as to be able to limitate the application of the resolution rule: the skip of a clause within a kernel deduction is the subset of its literals upon which the clause cannot be resolved further. Accordingly, the skip of every clause generated at the end of the k^{th} step of Tison's algorithm (i.e., once the variables x_1 ... x_k have been processed) contains all its literals built upon x_1 ... x_k while its kernel contains all the remaining literals. Thus, both the skip and the kernel of any clause within a kernel resolution proof depends on the literals that have been resolved so far to it. Given an A-ordering $<$, a literal l_2 built upon variable x_2 is considered larger than a literal l_1 built upon variable x_1 iff $x_1 < x_2$. l_1 and l_2 are considered equal iff $x_1 = x_2$. Every literal of an input clause belongs to its kernel. Now, two clauses γ_1 and γ_2 can be resolved upon literal l only if l belongs to the kernel of γ_1 and $\neg l$ to the kernel of γ_2, and the kernel of their resolvent consists of all its literals that are larger or equal to l. Interestingly, for any A-ordering, kernel resolution is complete in consequence finding (even in the first-order clausal case) [del Val, 1999] (we shall return to kernel resolution in Section 3.3.2).

Other resolution-based algorithms for computing prime implicates have been proposed in [Kean and Tsiknis, 1990b] [Jackson, 1992]. They aim at minimizing the number of resolvents to be generated and at deriving logically strongest resolvents first. Thus, the algorithm by Kean and Tsiknis [Kean and Tsiknis, 1990b] is an incremental algorithm for computing prime implicates that is a refinement of Tison's: given the set $PI(\Phi)$ of the prime implicates of some formula Φ and a clause Σ, the prime implicates of $(\Phi \wedge \Sigma)$ are computed. As any incremental algorithm for computing prime implicates, the algorithm by Kean and Tsiknis (and its optimizations) can be used to compute the set $PI(\Phi)$ from $\Phi = \gamma_1 \wedge \ldots \wedge \gamma_n$ by calling it $n-1$ times, so as to successively incorporating the clauses γ_2, ..., γ_n into the current set of prime implicates (equal to $PI(\{\gamma_1\}) = \{\gamma_1\}$ initially, then $PI(\{\gamma_1, \gamma_2\})$, and so on). Clearly enough, the ordering with which clauses of Φ are taken into account can have a great influence on the efficiency of the approach.

Within the algorithm by Kean and Tsiknis, starting with Σ as a top clause, every consensus that is performed uses one clause that has Σ as an ancestor and one clause of $PI(\Phi)$ [11]. Specifically, consensi between clauses of $PI(\Phi)$ (that can only generate implicates of Φ) are avoided. Additionally, the consensus operation is not applied in an iterative way over all the variables occurring in $Var(PI(\Phi)) \cup Var(\Sigma)$, but only over the variables of $Var(\Sigma)$ that appear with one sign in Σ and with the opposite sign in $PI(\Phi)$ (so-called *biform variables*); thus, considering an A-ordering over $Var(\Sigma)$ is sufficient.

Several optimized versions of the algorithm by Kean and Tsiknis also exist. In particular, *root optimization* [Kean and Tsiknis, 1990b] consists in checking whether there exists a clause π in $PI(\Phi)$ s.t. π and Σ can be resolved upon x (viewed as a literal) and the corresponding consensus Σ' subsumes Σ. In this case, it is valuable to consider any A-ordering for which x is in the first position so as to perform the consensus operation over x in priority (in some sense, this leads to replace the input clause Σ by the stronger one Σ').

Another interesting optimization is based on the notion of *restriction* [Kean, 1992]: for every clause γ that has been derived at some step, let its restriction be the set of variables computed by gathering all the variables of

[11] Accordingly, this algorithm can be considered as based on both the input resolution strategy [Chang, 1970], coupled with the set-of-support resolution one, with Σ as set of support [Wos *et al.*, 1967].

$Var(\Sigma)$ that have been resolved in the (input) resolution proof of γ, updated by removing every variable that does not belong to the restriction of any clause γ' that has been found subsumed by γ; then, resolving γ with any π of $PI(\Phi)$ s.t. the resolvent would contain a variable of the restriction of γ is useless [12].

Another source of improvement within the algorithm by Kean and Tsiknis consists in limiting the number of deduction checks that must be performed (thanks to local deduction checks, see [Kean and Tsiknis, 1990b] [Kean and Tsiknis, 1992] for details) and/or the efficiency of deduction checks. Thus, observing that the number of deduction checks between clauses grows as the square of the number of clauses in the current set, de Kleer [de Kleer, 1992] shows how to take advantage of a data structure (called a trie) for representing the current set of clauses so as to perform deduction checks more efficiently.

The algorithm by Jackson [Jackson, 1992] can be considered as an extension of the algorithm by Kean and Tsiknis in the sense that it takes as inputs both the set $PI(\Phi)$ of prime implicates of a formula Φ and a CNF formula Σ (and not a single clause). Based on the set-of-support resolution strategy (with Σ as set of support), resolution between clauses of $PI(\Phi)$ is avoided. However, contrariwise to Kean and Tsiknis', no fixed A-ordering is used in this algorithm but the strategy is guided dynamically by the clauses that are generated and their capacity to generate small resolvents. In order to minimize the size of the current set of clauses, deduction checks are performed whenever a consensus is generated: if subsumed by one of the current clauses, it is immediately removed (*forward subsumption*), otherwise it is added to this set and every other clause that is subsumed by the newly added one is removed from the set of current clauses (*backward subsumption*). In Jackson's algorithm, a current set of clauses - containing initially those of Σ - is updated by adding resolvents from clauses of the set with clauses from $PI(\Phi)$ and Σ. To each clause γ of the current set is associated the subset of $PI(\Phi) \cup \Sigma$ of all the clauses that can be resolved with γ so as to produce a consensus that is minimal w.r.t. logical deduction in the set of candidate resolvents (every clause that would generate a non-minimal resolvent when resolved with γ is not considered, this enables preempting forward subsumption). To each γ is also associated a score that indicates the number of literals of any smallest resolvent (in size) that can be generated using the clauses associated with γ. As an exception, any clause of the current set that can be resolved with a clause of Σ so as to generate a minimal resolvent is given a null score. The current set of clauses is sorted

[12]Note here that the simpler optimization called history restriction in [Kean and Tsiknis, 1990b] can lead to generate an incomplete set of prime implicates (see [Kean, 1992] for details). The problem is related to the fact that simple subsumed clauses deletion rules cannot be used within sophisticated linear resolution strategies like model elimination, withour questioning completeness.

according to this score (in increasing order). Thus, the first clause γ of the ordered set is one of those that may generate one of the shortest (hence minimal) resolvent (only a minimal resolvent if a productive resolution with a clause of Σ is possible). In Jackson's algorithm, such a clause is the first to be resolved. If it survives the forward deduction checks, the corresponding smallest resolvent is added to the current set of clauses. In $PI(\Phi)$, Σ and the current set of clauses, every clause implied by the added consensus is then removed (backward subsumption). The sets of clauses and scores attached to every clause of the current set are updated dynamically and this set is re-sorted dynamically.

To conclude with resolution-based techniques, let us mention that the sophisticated strategies that have been designed so far for computing prime implicates from a given stable production field $\mathcal{L}_T$ (especially, SOL-resolution [Inoue, 1992] and SFK-resolution [del Val, 1999], that are presented in the following) can also be used to generate prime implicates since the set of all clauses from $PROP_{PS}$ is a stable production field.

Beyond resolution techniques, many approaches to compute prime implicates are based on the structure of the input formula Σ. Unfortunately, in the general case, the prime implicates of a propositional formula cannot be characterized through the prime implicates of its subformulas in an efficient way. Of course, thanks to Corollary 3.2, the prime implicates of $\neg\Sigma$ (resp. of $\Phi \wedge \Sigma$) are also given by the prime implicates of the formula $\neg PI(\Sigma)$ (resp. of $PI(\Phi) \wedge PI(\Sigma)$) but this does not help a lot from a computational point of view.

However, some specific cases deserve specific comments. The first case concerns the prime implicates of $(\Phi \wedge \Sigma)$ whenever Σ is a literal (or, equivalently, a unit clause). Indeed, given that for every clause γ, $\Phi \wedge \Sigma \models \gamma$ holds iff $\Phi \models (\neg\Sigma \vee \gamma)$ holds, we have that γ is an implicate of $\Phi \wedge \Sigma$ iff there exists an implicate π_γ of Φ s.t. $\pi_\gamma \equiv (\neg\Sigma \vee \gamma)$. Accordingly, we have:

Proposition 3.7 *Let Φ be any formula from $PROP_{PS}$ and Σ be a literal from $PROP_{PS}$ that is not the boolean constant true. We have $PI(\Phi \wedge \Sigma) = min(\{\Sigma\} \cup \{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\} \cup \{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}, \models)$.*

This proposition can be considered to some extent as a by-product of the correctness of the algorithm by Kean and Tsiknis [Kean and Tsiknis, 1990b] for the incremental generation of prime implicates. Indeed, the set $min(\{\Sigma\} \cup \{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\} \cup \{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}, \models)$ is equivalent to the set of clauses that is obtained at the end of the execution of this algorithm.

Example 3.16 *Let $\Phi = (a \Rightarrow b) \wedge (b \vee c) \wedge (a \vee d) \wedge (b \Rightarrow e)$ and $\Sigma = a$.*

$PI(\Phi) = \{(\neg a \vee b), (b \vee c), (a \vee d), (b \vee d), (\neg b \vee e), (\neg a \vee e), (c \vee e), (d \vee e)\}$ *and* $PI(\Phi \wedge \Sigma) = \{a, b, e\}$.

Proposition 3.7 shows how the prime implicates of a conjunctive formula $\Phi \wedge \Sigma$ can be computed in an incremental way from the prime implicates of Φ when Σ is a literal. Interestingly, it shows that the number of prime implicates of $\Phi \wedge \Sigma$ cannot exceed the number of prime implicates of Φ by more than one. This deeply contrasts with the situation where Σ is a clause (or a unconstrained formula) (see Section 3.4). As to time complexity, implementing this proposition in a straightforward way already gives an algorithm for computing $PI(\Phi \wedge \Sigma)$ that runs in time polynomial in the size of $PI(\Phi)$ (a upper bound of the number of deduction checks to be performed is $(k+1)^2$ if Φ has k prime implicates). Clearly enough, some deduction checks can be avoided in the minimization step (we did not detail it inside the proposition for the sake of simplicity). Especially, no deduction checks are to be performed between Σ and the clauses of $\{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\}$, unless Φ is valid (in this case, $PI(\Phi \wedge \Sigma) = \{\Sigma\}$) or $\Phi \models \neg\Sigma$ holds (in this case, $PI(\Phi \wedge \Sigma) = \{false\}$). Note that checking whether or not Φ is valid (resp. $\Phi \wedge \neg\Sigma$ holds) can be done in time mainly polynomial in the size of $PI(\Phi)$. No deduction checks are to be performed between $\{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}$ and Σ unless Φ is contradictory (this can be detected in a very efficient way from $PI(\Phi)$ and, and in this case, we have $PI(\Phi \wedge \Sigma) = \{false\}$). No deduction checks are to be performed between $\{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}$ and $\{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\}$, and no deduction checks between clauses from $\{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}$ (resp. $\{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\}$) are required. Finally, the case where Σ is the boolean constant *true* can be easily handled in a preliminary step (in this case, $PI(\Phi \wedge \Sigma) = PI(\Phi)$).

Clearly enough, Proposition 3.7 can be extended so as to compute the prime implicates of $PI(\Phi \wedge \Sigma)$ from the prime implicates of Φ and a term Σ. Indeed, it is sufficient to compute the prime implicates of $\Phi \wedge l$ in an iterative way for every literal l of Σ, considering the resulting set of each step as the starting one of the next step. As a straightforward consequence, the number of prime implicates of $\Phi \wedge \Sigma$ when Σ is a term cannot exceed $k + m$, where k is the number of prime implicates of Φ and m is the number of literals occurring in Σ. Moreover, the total number of deduction checks to be performed cannot exceed $\sum_{i=1}^{m}(k+i)^2$ (which is in $\mathcal{O}(k^2)$ when m is considered as a constant). Accordingly, the algorithm sketched above allows one to compute the prime implicates of $\Phi \wedge \Sigma$ from the prime implicates of Φ and a term Σ in time polynomial in the input size.

In the following, Φ-prime implicates will be characterized through prime implicates. Conversely, the prime implicates of a conjunctive formula $\Phi \wedge \Sigma$ can also be characterized through the Φ-prime implicates of Σ. Intuitively, the prime implicates of $\Phi \wedge \Sigma$ are among the logically strongest clauses from the prime implicates of Φ and the Φ-prime implicates of Σ. To be more precise:

Proposition 3.8 *Let Φ and Σ be formulas from $PROP_{PS}$. $PI(\Phi \wedge \Sigma) = min(PI(\Phi) \cup PI^{\Phi}(\Sigma), \models)$ (up to logical equivalence).*

Example 3.17 *Let $\Phi = (a \Rightarrow b) \wedge (b \vee c) \wedge (a \vee d) \wedge (b \Rightarrow e)$ and $\Sigma = a$. $PI(\Phi \wedge \Sigma) = min(PI(\Phi) \cup PI^{\Phi}(\Sigma), \models) = min(\{(\neg a \vee b), (b \vee c), (a \vee d), (b \vee d), (\neg b \vee e), (\neg a \vee e), (c \vee e), (d \vee e)\} \cup \{a, b, e\}, \models) = \{a, b, e\}$.*

Clearly enough, many deduction checks that a naive implementation of Proposition 3.8 would suggest can be avoided; especially, no deduction checks are to be performed inside $PI(\Phi)$ and inside $PI^{\Phi}(\Sigma)$. Moreover, no Φ-prime implicate of Σ can be entailed by a prime implicate of Φ.

This characterization enables taking advantage of "efficient" strategies for computing Φ-prime implicates (especially, SOL-resolution [Inoue, 1991] [Inoue, 1992]) so as to compute prime implicates.

Another valuable characterization of prime implicates is obtained whenever the knowledge base Σ is a disjunctive formula. In this case, the following *distribution property* can prove helpful:

Proposition 3.9 (distribution) *Let $\Sigma_1, \ldots, \Sigma_k$ be formulas from $PROP_{PS}$. $PI(\Sigma_1 \vee ... \vee \Sigma_k) = min(\{(\pi_{\Sigma_1} \vee ... \vee \pi_{\Sigma_k}); \pi_{\Sigma_1} \in PI(\Sigma_1), ..., \pi_{\Sigma_k} \in PI(\Sigma_k)\}, \models)$.*

Example 3.18 *Let $\Sigma_1 = (a \vee b) \wedge c$ and $\Sigma_2 = (\neg a \vee b) \wedge (b \vee c)$. $PI(\Sigma_1) = \{(a \vee b), c\}$ and $PI(\Sigma_2) = \{(\neg a \vee b), (b \vee c)\}$. Accordingly, we have $PI(\Sigma_1 \vee \Sigma_2) = min(\{(a \vee b \vee \neg a \vee b), (a \vee b \vee b \vee c), (c \vee \neg a \vee b), (c \vee b \vee c)\}, \models) = \{(b \vee c)\}$ (up to logical equivalence).*

This property is sometimes called *multiplication* [Socher, 1991], *model-crossing* [Boï *et al.*, 1992] or *duality* [Castell, 1997]. Distribution, originally proposed by Nelson in 1955 [Nelson, 1955], is the key property on which many algorithms for computing prime implicants/implicates are based, e.g., [Slagle *et al.*, 1970] [Jackson and Pais, 1990] [Socher, 1991] [Ngair, 1993] [Ramesh and Murray, 1993] [Castell, 1996] [Ramesh *et al.*, 1997] [Shiny and Pujari, 1998] [Shiny and Pujari, 1999]. Indeed, this property shows how the prime implicates of a disjunction can be derived through the prime implicates of its disjuncts, by selecting the logically strongest clauses among those obtained by taking the disjunction of one prime implicate per disjunct. Computing the latter set comes down to distributing the "or" connective over the "and" connective (that implicitly gathers the prime implicates in their set), just like the multiplicative operator for real numbers distributes over the additive one.

In contrast to the resolution-based approaches for computing prime implicates, the distribution-based ones cannot be extended to the first-order case in general (the reason is that a clause is a universally quantified disjunction, and universal quantification does not distribute over disjunction in the general case).

While Proposition 3.7 shows how to compute $PI(\Phi \wedge \Sigma)$ from $PI(\Phi)$ and a term Σ in an efficient way, Proposition 3.9 does not show that $PI(\Sigma_1 \vee ... \vee \Sigma_k)$ can be computed efficiently from the sets $PI(\Sigma_i)$ $(i \in 1 .. k)$ in the general case. Indeed, $PI(\Sigma_1 \vee ... \vee \Sigma_k)$ can contain up to $card(PI(\Sigma_1)) * ... * card(PI(\Sigma_k))$ clauses, which clearly is exponential in k in the worst case (just consider the case where every literal in $PI(\Sigma_1) \cup ... \cup PI(\Sigma_k)$ occurs only once in this set and every clause of this set contains at least two literals: no minimization is possible). We shall return later to this issue (see Section 3.4).

Of course, this is the worst case. In practice, some minimization checks succeed, allowing candidates to be removed. As suggested by the above example, when implementing the distribution property so as to generate the prime implicates of a disjunction, one of the main difficulty is to avoid generating some clauses (as the disjunctions of two prime implicates, one per disjunct) that are removed afterwards because they are not minimal w.r.t. logical deduction. Indeed, both the time needed to generate these clauses, to compare them with all the other clauses so as to perform the minimization step w.r.t. $\models$, and the space required to store them can be so huge that a naive implementation of the property would make the approach impractical. The following corollary contributes to circumvent this difficulty:

Corollary 3.3 *Let Σ_1 and Σ_2 be two formulas from $PROP_{PS}$. If π_{Σ_1} is a prime implicate of Σ_1 s.t. there exists a prime implicate π_{Σ_2} of Σ_2 satisfying $\pi_{\Sigma_2} \models \pi_{\Sigma_1}$, then π_{Σ_1} is a prime implicate of $\Sigma_1 \vee \Sigma_2$.*

In particular, this corollary shows that every prime implicate of Σ_1 that also is a prime implicate of Σ_2 is a prime implicate of their disjunction. Actually, efficient implementations of the distribution property for computing $PI(\Sigma_1 \vee \Sigma_2)$ first compute the intersection $PI(\Sigma_1) \cap PI(\Sigma_2)$, while removing it from both $PI(\Sigma_1)$ and $PI(\Sigma_2)$, and then add to the current set of prime implicates every clause concerned by the corollary above (and remove it from its original set) [13]. It is then sufficient to apply the distribution property to the simplified sets, while avoiding to derive tautologies (which can be easily done when clauses are indexed by the literals occurring in them). Finally, minimality w.r.t. $\models$ is checked each time a new clause is generated so as to keep at each step only minimal clauses. For the example above, taking

[13] We assume here without any loss of generality that clauses are represented in a canonical way, i.e., two equivalent clauses share the same representation.

advantage of this strategy would lead to generate only half of the clauses that would be derived otherwise.

Some additional heuristics consist in considering first clauses from $PI(\Sigma_1)$ and $PI(\Sigma_2)$ of smallest size, or those containing the literals occurring the most frequently in their respective sets. They can be used to tentatively generate first the minimal elements (w.r.t. $\models$) of $PI(\Sigma_1 \vee \Sigma_2)$, allowing non-minimal clauses to be removed as soon as they are generated.

Though it is the case in some of its implementations (e.g., [Slagle *et al.*, 1970]), the distribution property does not require to generate in one step the whole set of clauses generated by disjoining one clause per set $PI(\Sigma_i)(i \in 1 \,..\, k)$. Instead, prime implicates can be generated incrementally by considering only two sets of prime implicates at a time, generating the set of prime implicates of their disjunction, and replacing the former two sets by the latter. Since minimizations are performed each time the prime implicates of a disjunction are computed, many clauses might be removed at each step. This prevents them from being disjoined further, as a one-step application of the distribution property would require. Such an incremental handling (that can be found in several implementations, e.g., [Jackson and Pais, 1990]) also calls for heuristics to gather sets of prime implicates together. Roughly, these heuristics aim at promoting the production of logically strongest clauses first through the distribution property.

Combining Propositions 3.7 and 3.9 gives an algorithm for computing $PI(\Phi \wedge \Sigma)$ from $PI(\Phi)$ and a DNF formula Σ. Indeed, if $\{\gamma_1, ..., \gamma_k\}$ is the set of terms occurring in Σ, then we have $(\Phi \wedge \Sigma) \equiv (\Phi \wedge \gamma_1) \vee ... \vee (\Phi \wedge \gamma_k)$; accordingly, $PI(\Phi \wedge \Sigma)$ can be computed taking advantage of Proposition 3.9 where each $PI(\Phi \wedge \gamma_i)$ $(i \in 1 \,..\, k)$ has been computed in a preliminary step from $PI(\Phi)$ and γ_i, thanks to Proposition 3.7. To some extent, this combination can be considered as the formal ground of Ngair's algorithm [Ngair, 1993] for the incremental prime implicates generation for conjunctions of DNF formulas Σ_1, ..., Σ_k. Within this algorithm, $PI(\Sigma_1 \wedge \Sigma_2 \wedge ... \wedge \Sigma_k)$ is computed as $PI(PI(...(PI(true \wedge \Sigma_1) \wedge \Sigma_2) \wedge ... \wedge \Sigma_k)$. Of course, there is no guarantee that the size of the output is polynomially related to the input size in the general case.

Example 3.19 *Let* $\Phi = (a \Rightarrow b)$ *and* $\Sigma = (\neg b \vee c)$. $PI(\Phi \wedge \neg b) = \{\neg b, \neg a\}$ *and* $PI(\Phi \wedge c) = \{(\neg a \vee b), c\}$. *We have* $PI(\Phi \wedge \Sigma) = min(\{(\neg b \vee \neg a \vee b), (\neg b \vee c), (\neg a \vee \neg a \vee b), (\neg a \vee c)\}, \models) = \{(\neg b \vee c), (\neg a \vee c), (\neg a \vee b)\}$ *(up to logical equivalence).*

Another interesting corollary of Proposition 3.9 is obtained when Σ is given in *reduced DNF*. Indeed, an interesting feature of any reduced DNF is that, for every term γ in it, the literals occurring in γ are the prime implicates of γ (this explains the specific treatment of boolean constants and unsatisfiable terms, see Section 2.1). Moreover, considering that every term is represented by the set of its literals, we have:

Corollary 3.4 *Let* $\{\gamma_1, \ldots, \gamma_k\}$ *be a reduced DNF of* Σ. $PI(\Sigma) = min(\{(\pi_1 \vee \ldots \vee \pi_k); \pi_1 \in \gamma_1, \ldots, \pi_k \in \gamma_k\}, \models)$ *(up to logical equivalence).*

Example 3.20 *Let* $\Sigma = (a \vee (b \wedge c)) \wedge (\neg b \vee c)$. *The set* $\{(a \wedge \neg b), (a \wedge c), (b \wedge c)\}$ *is a reduced DNF of* Σ. *Thus,* $PI(\Sigma) = min(\{(a \vee a \vee b), (a \vee a \vee c), (a \vee c \vee b), (a \vee c \vee c), (\neg b \vee a \vee b), (\neg b \vee a \vee c), (\neg b \vee c \vee b), (\neg b \vee c \vee c)\}, \models) = \{(a \vee b), (a \vee c), (\neg b \vee c)\}$ *(up to logical equivalence).*

Clearly, strategies that prove efficient for the generation of the prime implicates of a disjunction also apply to the specific case where the sets of prime implicates under consideration are sets of literals. Especially, Corollary 3.3 consists here in extracting first the literals occurring in both sets as prime implicates of their disjunction, before applying the distribution property to the simplified sets. Moreover, heuristics can be used to gather terms together. Thus, on the example above, applying first the distribution property to terms $(a \wedge \neg b)$ and $(a \wedge c)$ proves computationally better than applying it first to terms $(a \wedge \neg b)$ and $(b \wedge c)$.

The distribution property for reduced DNF formulas can be implemented in several different ways. Thus, the algorithm by Slagle, Chang and Lee [Slagle *et al.*, 1970] consists in searching an ordered tree in a depth-first fashion in order to generate the prime implicates of a (reduced) DNF formula Σ. In this tree, the ordered set of branches immediately below a given node corresponds to the literals appearing in the increasing order of their occurrence frequency in the terms that remain to be visited at that node. In other words, the distribution property is performed in priority over the largest sets of terms sharing a literal. Nodes are labelled with sets of clauses, obtained by propagating in Σ the truth values of the selected literals from the root of the tree up to its nodes, and by removing in the resulting sets of clauses all the literals that precede the lastly selected literal (this prevents from considering branches that differ only by the ordering of their literals). Leaves are either success leaves (labelled with the empty set of clauses) or failure leaves (whenever every literal that could be elected is the negation of a literal that has already been elected in the curent branch, or has already appeared on a path to the left of the current node). The branches leading to success leaves are implicates of Σ and all the prime implicates of Σ are such leaves. Accordingly, an additional minimization step over these branches enables the prime implicates of Σ to be generated.

The algorithm by Socher [Socher, 1991] is close to the algorithm by S-lagle, Chang and Lee but it is based on on Bibel's connection method (or matrix method) [Bibel, 1981] [Bibel, 1982], a well-known automated deduction technique. In propositional matrix methods, a CNF (resp. DNF) formula Σ is represented by a boolean matrix M_Σ, indexed by $Lit(\Sigma)$ (the rows) and by the clauses (resp. terms) of Σ (the columns). Thus, for every literal l of $Lit(\Sigma)$ and every clause (or term) γ of Σ, $M_\Sigma(l, \gamma) = 1$ iff

l is a literal of γ. The matrix methods rely on the idea that every DNF (resp. CNF) of an inconsistent (resp. valid) formula contains inconsistent terms (resp. valid clauses), only. However, they do not generate explicitly a DNF (resp. CNF) of the given CNF (resp. DNF) formula. Instead, they do graph searching from it, enumerating its paths, where a path consists in one literal from each clause or each term. If every path is not fundamental (i.e. it contains a complementary pair of literals), the formula is inconsistent or valid (depending on the nature of the input - CNF formula versus DNF formula). Because a given complementary pair of literals often appears in more than one path, it permits clever search to avoid the explicit enumeration of all paths. Clearly enough, the set of all unsubsumed fundamental paths of a reduced DNF formula represent the set of its prime implicates. In the algorithm by Socher, the search through the set of paths of a reduced DNF formula is guided by the occurrence frequency of literals (like in the algorithm by Slagle, Chang and Lee). In addition, once a literal l has been processed (i.e., all the paths corresponding to the prime implicates of Σ containing l have been generated, possibly among other such implicates), the current matrix is updated by removing the row l (logically, l is forgotten within Σ, cf. Section 3.3.2) and maximization checks are performed so as to keep in the matrix only the colums that correspond to maximal terms w.r.t. $\models$. This simplification step is a valuable improvement over the algorithm by Slagle, Chang and Lee. Recently, Socher's algorithm has been improved by Shiny and Pujari [Shiny and Pujari, 1998] [Shiny and Pujari, 1999]. These authors suggest a divide-and-conquer strategy for computing the prime implicates of a reduced DNF formula Σ; this strategy consists in focusing (in an independent and recursive way) on the prime implicates of Σ containing a literal l and on those not containing l (where l is a literal occurring in Σ). This approach is actually very close to the prime implicates algorithms based on Shannon decomposition that are presented in the following.

The algorithm by Jackson and Pais [Jackson and Pais, 1990] is essentially a breadth-first search through the matrix of a (reduced) DNF formula Σ for computing all its fundamental unsubsumed paths. However, the search is not guided by the literals occurring in Σ by but its terms. To be more precise, all the terms of Σ are ordered according to their size, the shortest ones first. Once one of the shortest terms has been selected, every literal in it is considered as the beginning of a path. At each subsequent step, one of the remaining terms is selected so as to maximize the product of the number of literals of the term that have already been considered with the number of literals of the terms whose negations have already been considered. The purpose is to promote the generation of the logically strongest implicates of Σ. Then the distribution property is applied. Thus, the current paths are updated with the literals of the selected term; each time a path shares a literal with the selected term, no extension of that path is necessary (it already corresponds to an implicate of that term); furthermore, each time a path contains the negation of a literal of the selected term, no extension of the path with that literal is necessary (it would give rise to a tautology). Finally, every extended path that is a superset of a path that has not been extended can be removed (this is a minimization step, performed each time a new term is considered). Once all the terms of Σ have been processed, the resulting set of paths corresponds to the set of prime implicates of Σ.

The algorithm by Jackson and Pais can be extended in several ways. In [Marquis, 1991], the distribution step is not performed sequentially between a current set of paths (a CNF formula) and a selected term (a particular CNF) but between two selected sets of paths. A heuristic is presented so as to find out the CNFs (including the original terms) on which the distribution property should be applied in priority. Thus, several sets of paths can be first formed in parallel and combined later. Another extension concerns the nature of the input formula. Indeed, the connection method also is applicable to NNF formulas (it is only necessary to refine the notion of a path). Ramesh and Murray [Ramesh and Murray, 1993] [Ramesh *et al.*, 1997] give an algorithm for computing prime implicates for NNF formulas. This extension of Jackson and Pais' algorithm can take advantage of an efficient inference method, called path dissolution [Murray and Rosenthal, 1993]. It mainly consists in restructuring the input formulas s.t. all paths through a selected link are eliminated. Even if the formula starts out in CNF, a single dissolution step produces an unnormalized formula.

The algorithm by Castell [Castell, 1996] [Castell, 1997] for computing prime implicates is based on the Davis/Logemann/Loveland search algorithm [Davis *et al.*, 1962], on which the most efficient algorithms for checking the satisfiability of a CNF formula (the SAT problem) rely. At each step, a literal of Σ is selected and it gives rise to two branches of the search tree: the first one corresponds to formulas containing l, while the other one corresponds to formulas not containing l. The Davis/Logemann/Loveland search algorithm is used twice. The first one aims at generating a DNF of Σ. The second one enables computing the prime implicates of Σ by observing that every implicate of Σ and every term of a reduced DNF of Σ necessarily share at least one literal, and for every prime implicate of Σ, there is a term in the reduced DNF that has exactly one literal in common with this prime implicate (we shall return to this issue in Section 3.4). The practical interest of the approach is based on the existence of "efficient" branching rules and propagation rules (especially, unit propagation)[14] and the fact that no deduction checks between clauses are necessary. This contrasts deeply with resolution-based algorithms for which most of the computation time is consumed performing such deduction checks. Castell's algorithm is particularly efficient in the case where Σ is inconsistent (this situation is recognized once

[14]However, as explained previously, the pure literal deletion rule that is commonly used in algorithms for SAT based on Davis/Logemann/Loveland's search cannot be used here.

the first search has been performed, then $PI(\Sigma) = \{false\}$ can be returned immediately) or when the size of the DNF of Σ computed at the first step remains small enough. Interestingly, the algorithm by Castell also has an anytime flavor: once a DNF of Σ has been computed (end of the first step), the search algorithm can be interrupted at any time during the second step, generating a subset of $PI(\Sigma)$.

Another valuable characterization of the prime implicates of a formula Σ can be achieved whenever Σ is given by a Shannon expansion.

Definition 3.13 (Shannon expansion) *Let Σ be a formula from $PROP_{PS}$ and x be a variable from PS. The Shannon expansion of Σ around x is the formula $(\neg x \wedge \Sigma_{x \leftarrow false}) \vee (x \wedge \Sigma_{x \leftarrow true})$.*

Clearly enough, every Shannon expansion of Σ is equivalent to Σ. Exploiting the distribution property, we derive the following recursive characterization of the prime implicates of Σ:

Proposition 3.10 *Let Σ be a formula from $PROP_{PS}$ and x a variable from PS. Let $PI^*(\Sigma)$ be $PI(\Sigma)$ whenever Σ is not valid, $\emptyset$ otherwise. We have:*
$PI^*((\neg x \wedge \Sigma_{x \leftarrow false}) \vee (x \wedge \Sigma_{x \leftarrow true}) = PI^*(\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true})$
$\cup\{(\neg x \vee \pi_{\Sigma_{x \leftarrow true}}); \pi_{\Sigma_{x \leftarrow true}} \in PI^*(\Sigma_{x \leftarrow true})$
and $\forall \pi_{\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true}} \in PI^*(\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true}), \pi_{\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true}} \not\models \pi_{\Sigma_{x \leftarrow true}}\}$
$\cup\{(x \vee \pi_{\Sigma_{x \leftarrow false}}); \pi_{\Sigma_{x \leftarrow false}} \in PI^*(\Sigma_{x \leftarrow false})$
and $\forall \pi_{\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true}} \in PI^*(\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true}), \pi_{\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true}} \not\models \pi_{\Sigma_{x \leftarrow false}}\}$.

Clearly enough, we have $PI(\Sigma) = \{true\}$ whenever $PI^*(\Sigma) = \emptyset$, $PI(\Sigma) = PI^*(\Sigma)$ otherwise.

Example 3.21 *Let $\Sigma = ((a \wedge b) \vee (a \wedge c)) \vee (\neg a \wedge (\neg b \vee c))$. The Shannon expansion of Σ around a is equivalent to $(\neg a \wedge (\neg b \vee c)) \vee (a \wedge (b \vee c))$. $PI(\Sigma_{a \leftarrow false}) = \{(\neg b \vee c)\}$ and $PI(\Sigma_{a \leftarrow true}) = \{(b \vee c)\}$. Accordingly, $PI(\Sigma_{a \leftarrow false} \vee \Sigma_{a \leftarrow true}) = \{(\neg b \vee c \vee b \vee c)\}$, hence $PI^*(\Sigma_{a \leftarrow false} \vee \Sigma_{a \leftarrow true}) = \emptyset$. We also have $PI(\Sigma) = PI^*(\Sigma) = \{(\neg a \vee b \vee c), (a \vee \neg b \vee c)\}$.*

Actually, Proposition 3.10 can be viewed as a corollary of Proposition 3.9. Indeed, since x neither occurs in $\Sigma_{x \leftarrow false}$ nor in $\Sigma_{x \leftarrow true}$, the prime implicates of $(\neg x \wedge \Sigma_{x \leftarrow false})$ (resp. $(x \wedge \Sigma_{x \leftarrow true})$) are $\{\neg x\} \cup PI(\Sigma_{x \leftarrow false})$ (resp. $\{x\} \cup PI(\Sigma_{x \leftarrow true})$) whenever $\Sigma_{x \leftarrow false}$ neither is valid, nor contradictory (this can be considered as a by-product of Proposition 3.7). Then, the distribution property is used between both sets of prime implicates but useless deduction checks are avoided; indeed, a clause containing x (resp. $\neg x$) cannot be logically stronger than a clause not containing x (resp. $\neg x$) –especially, those containing $\neg x$ (resp. x)– unless the latter is a tautology (which is handled by computing PI^* instead of PI). Moreover, no deduction checks may succeed between two different clauses from $\{(x \vee \pi_{\Sigma \leftarrow false}); \pi_{\Sigma \leftarrow false} \in$

$PI^*(\Sigma_{x \leftarrow false})\}$ (resp. $\{(\neg x \vee \pi_{\Sigma \leftarrow true}); \pi_{\Sigma \leftarrow true} \in PI^*(\Sigma_{x \leftarrow true})\}$). In any efficient implementation of Proposition 3.10 (like the one described in [Madre and Coudert, 1991]), $PI^*(\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true})$ is the first set to be computed once both $PI^*(\Sigma_{x \leftarrow false})$ and $PI^*(\Sigma_{x \leftarrow true})$ have been derived (this is a way to promote backward subsumption and to maintain as small as possible the current set of clauses). For similar reasons, as explained previously, the set $PI^*(\Sigma_{x \leftarrow false}) \cap PI^*(\Sigma_{x \leftarrow true})$ is derived first in the computation of $PI^*(\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true})$, and every clause from this intersection is removed from both $PI^*(\Sigma_{x \leftarrow false})$ and $PI^*(\Sigma_{x \leftarrow true})$ before the computation of the two remaining sets of clauses. Note that, since both $\Sigma_{x \leftarrow false}$ and $\Sigma_{x \leftarrow true}$ are equivalent to Σ when Σ is independent from x, the proposition above trivializes when the Shannon expansion of Σ around a variable it is independent from which is used; indeed, it comes down to $PI^*(\Sigma) = PI^*(\Sigma_{x \leftarrow false} \vee \Sigma_{x \leftarrow true}) = \ldots PI^*(\Sigma)$. Nevertheless, such trivialization shows the proposition true when Σ is valid or contradictory, since in this situation, Σ is independent from every literal.

The algorithm proposed by Madre and Coudert [Madre and Coudert, 1991] is mainly based on Proposition 3.10. In a first step, a compact representation of a Shannon tree of Σ is computed under the form of a BDD (Binary Decision Diagram) [15] [Akers, 1978]. Identical subformulas occurring in such a tree are physically shared; computing the prime implicates of Σ from the prime implicates of its subformulas in a bottom–up way, this prevents from computing twice or more the prime implicates of every subformula occurring more than once in the tree. Note that, from a practical point of view, a BDD of Σ is not generated from a Shannon tree of Σ that would have been computed first; indeed, the size of this tree is exponential in the number of variables occurring in Σ in the worst case. Instead, a BDD of Σ is computed in a recursive way from the BDDs of its subformulas [Bryant, 1986]. Indeed, a BDD of Σ can be computed in time polynomial in the size of the BDD of its direct subformula(s). Obviously, this does not question the worst case complexity (since it can be the case that no space reduction is achievable when computing the BDD). However, this technique appears empirically viable in many situations where computing a Shannon tree of Σ is out of possible. Then, Proposition 3.10 is used to recursively compute the prime implicates of Σ, starting from the sinks of the BDD (usually the nodes $true$ and $false$). Note here that every literal Σ is independent from which is removed from its BDD representation since every restriction formula of Σ is independent from variable x whenever Σ is independent from x. Especially, any BDD of a valid (resp. contradictory) formula contains one node only: the node $true$ (resp. the node $false$). Clearly enough, the conciseness of the BDD representation of Σ has a major impact on the efficiency of the

[15] Actually, typed decision graphs, a very concise BDD-like representation of propositional formulas, are used.

algorithm. Several variable orderings can be used giving rise to BDDs whose size can heavily differ. Some heuristics for finding out a "good" variable ordering are available [Friedman and Supowit, 1990], but there is no efficient algorithm to determine one that minimizes the size of the BDD (this task is NP-hard). In practice, a BDD-based computation of prime implicates is particularly efficient when Σ has many prime implicates; it is particularly inefficient when Σ has no prime implicate, except the empty clause [Uribe and Stickel, 1994].

3.3.2 L–prime implicates

Let us now focus on the metatheoretic properties of L-prime implicates. As a straightforward consequence of their definition, L–prime implicates are prime implicates; to be more precise, the L–prime implicates of Σ are the prime implicates of Σ that are built up from L. Formally:

Proposition 3.11 *Let Σ be a formula from $PROP_{PS}$ and L be a finite, non-empty subset of L_{PS}. $PI_L(\Sigma) = \{c \in PROP_{PS}/\equiv; Lit(c) \subseteq L$ and $(\exists \pi \in PI(\Sigma))(c \equiv \pi)\}$.*

Example 3.22 *Let $L = \{\neg a, b\}$ and $\Sigma = (a \Rightarrow b) \wedge (b \Rightarrow c)$. $PI(\Sigma) = \{(\neg a \vee b), (\neg b \vee c), (\neg a \vee c)\}$ and $PI_L(\Sigma) = \{(\neg a \vee b)\}$.*

Indeed, since the set of clauses built up from L always is a stable production field, the notions of prime L-implicate and L-prime implicate coincide. Especially, if π is a L-prime implicate of Σ, then no prime implicate π_Σ of Σ s.t. $\pi_\Sigma \models \pi$ may exist, unless $\pi_\Sigma \equiv \pi$.

As evoked previously, such a characterization of the logically strongest implicates from a given production field in terms of *selected* prime implicates can be reached whenever the production field is stable. Interestingly, this suggests a way to compute them (as soon as the production field is a recursive set, which is a reasonable assumption). Thus, $PI_L(\Sigma)$ can be generated by filtering out from $PI(\Sigma)$ every clause π for which no clause containing only literals from L and equivalent to π exists. Note here that a purely syntactic filtering cannot be adequate since a clause that is not built up from L can belong to the same equivalence class than a clause built up from L (just consider tautologies). The extreme case where $L = \emptyset$ also deserves some specific comments, since it is not covered by Proposition 3.11. Indeed, the subset of clauses containing only literals from $\emptyset$ contains two clauses (up to logical equivalence), namely $true$ and $false$ (also represented by the empty clause). If $false$ is an implicate of Σ, then Σ is contradictory and $false$ is the unique prime implicate of Σ; in this situation, Proposition 3.11 holds. In the remaining case, $true$ is the sole element of $PI_\emptyset(\Sigma)$ but no equivalent clause can be found in $PI(\Sigma)$, except when Σ is valid. In any case, the filtering process suggested by Proposition 3.11 can be accomplished in time polynomial in the size of $PI(\Sigma)$ since checking whether a clause is a tautology is computationally easy. This shows that the computation of $PI_{\emptyset}(\Sigma)$ from $PI(\Sigma)$ can be achieved in an efficient way.

Proposition 3.11 also shows how algorithms for computing prime implicates of CNF formulas can be used to derive the prime implicates of an unrestricted formula Σ without turning it first into CNF in a "classical way" (the one requiring exponential space in the worst case). Instead, Σ is first turned into CNF introducing new symbols (and normalization requires polynomial time, only), then the prime implicates of the resulting CNF formula Σ' are computed. From the prime implicates of Σ', the prime implicates of the original Σ can be filtered out by keeping only the clauses that do not contain any of the new symbols introduced during the normalization step. Nevertheless, there is no guarantee that this technique is computationally better than the one achieved by turning Σ into CNF without introducing any new symbol; indeed, it can be the case that Σ' has exponentially more prime implicates than the corresponding Σ [Ngair, 1993].

Another valuable consequence of Proposition 3.11 is that the notion of L–prime implicate can be seen as a generalization of the notion of prime implicate, where the latter is recovered when $L = L_{PS}$[16]. Together with Corollary 3.1, Proposition 3.11 shows that $PI_L(\Sigma)$ always is a finite set. Moreover, $PI_L(\Sigma)$ can be empty (unlike $PI(\Sigma)$). Conversely, it is important to note that the L–prime implicates of Σ are the (L_{PS}-)prime implicates of some formula. Indeed, a characterization of the L-prime implicates of Σ is reached by focusing on the most general consequence of Σ that is built up from L. Such a formula can be generated by making Σ independent from every literal from $Lit(\Sigma)$, except those of L.

Now, in order to make Σ independent from L, we need the following notion of *literal forgetting* (more about it is to be found in [Lang *et al.*, 1999]):

Definition 3.14 (literal forgetting) *Let Σ be a formula from $PROP_{PS}$ and L a finite subset of L_{PS}. $forget(\Sigma, L)$ is the formula inductively defined as follows:*

- $forget(\Sigma, \emptyset) = \Sigma$,
- $forget(\Sigma, \{x\}) = \Sigma_{x \leftarrow true} \vee (\neg x \wedge \Sigma_{x \leftarrow false})$,
- $forget(\Sigma, \{\neg x\}) = \Sigma_{x \leftarrow false} \vee (x \wedge \Sigma_{x \leftarrow true})$,
- $forget(\Sigma, \{l\} \cup L) = forget(forget(\Sigma, L), \{l\})$

This definition is sound in the sense that the ordering in which literals of L are considered does not matter (this can be proved easily by induction on $card(L)$).

[16] As discussed before, $L = Dep(\Sigma)$ is sufficient.

Example 3.23 *Let* $\Sigma = (\neg a \vee b) \wedge (a \vee c) \wedge (b \vee c \vee d)$ *and* $L = Lit(\Sigma) \setminus \{a, b, c\} = \{\neg a, d\}$. *We have* $forget(\Sigma, L) \equiv (c \vee (a \wedge b))$.

By a straightforward induction, it is easy the prove that forgetting any set of literals within a satisfiable term simply consists in removing every occurrence of every literal from the set within the term.

Example 3.24 *Let* $\gamma = (\neg a \wedge b \wedge c)$. *We have* $forget(\gamma, \{\neg a, b\}) \equiv c$ *and* $forget(\gamma, \{a, b\}) \equiv (\neg a \wedge c)$.

Interesingly, forgetting within a DNF formula is tractable. Indeed, since the "or" connective is associative and commutative, it is easy to show that for every pair of formulas Σ_1 and Σ_2 of $PROP_{PS}$ and every finite subset L of L_{PS}, we have $forget(\Sigma_1 \vee \Sigma_2, L) \equiv (forget(\Sigma_1, L) \vee forget(\Sigma_2, L))$ (see [Lang *et al.*, 1999] for details). As a consequence, a formula equivalent to $forget(\Sigma, Lit(\Sigma) \setminus L)$ can be efficiently generated when Σ is a DNF formula by removing each contradictory term in it, then removing from each remaining term of Σ all the literals from $Lit(\Sigma) \setminus L$; the resulting formula is a DNF of $forget(\Sigma, Lit(\Sigma) \setminus L)$ and it can be generated in time polynomial in the input size.

By the way, it is important to note that such a decomposition property cannot be achieved in the general case for conjunctive formulas. Indeed, while $forget(\Sigma_1, L) \wedge forget(\Sigma_2, L)$ always is a logical consequence of $forget(\Sigma_1 \wedge \Sigma_2, L)$, the converse typically does not hold.

From a logical point of view, $forget(\Sigma, L)$ contains all the pieces of knowledge of Σ that are independent from L.

Proposition 3.12 $forget(\Sigma, L)$ *is the most general consequence of* Σ *that is independent from* L *(up to logical equivalence).*

Such a notion of forgetting generalizes the notion of *variable elimination* in propositional logic (that has been generalized to the first-order case by Lin and Reiter [Lin and Reiter, 1994]). Variable elimination is an important notion for consequence finding. It can be found in the literature under various names, like projection or marginalization (see e.g. [Kohlas *et al.*, 1999b]). Actually, variable elimination is about *variable forgetting*, i.e., the one achieved by not considering the literals signs; accordingly, the elimination of x in Σ can be defined by the quantified boolean formula $(\exists x)\Sigma \equiv forget(\Sigma, \{x, \neg x\})$.

Incidentally, it is worth noting that variable elimination enables a more concise description of Proposition 3.10: $PI((\neg x \wedge \Sigma_{x \leftarrow false}) \vee (x \wedge \Sigma_{x \leftarrow true})) = PI(\exists x \Sigma) \cup \{(x \vee \pi) | \pi \in PI^{\exists x \Sigma}(\Sigma_{x \leftarrow false})\} \cup \{(\neg x \vee \pi) | \pi \in PI^{\exists x \Sigma}(\Sigma_{x \leftarrow true})\}$.

We are now ready to characterize L–prime implicates as the prime implicates of a formula:

Proposition 3.13 *Let Σ be a formula from $PROP_{PS}$ and L be a finite subset of L_{PS}. $PI_L(\Sigma) = PI(forget(\Sigma, Lit(\Sigma) \setminus L))$.*

Example 3.25 *Let $\Sigma = (\neg a \vee b) \wedge (a \vee c) \wedge (b \vee c \vee d)$ and $L = Lit(\Sigma) \setminus \{a, b, c\}$. $PI_L(\Sigma) = PI(forget(\Sigma, Lit(\Sigma) \setminus L)) = \{(a \vee c), (b \vee c)\}$.*

Thanks to Proposition 3.13, we can easily identify the formulas that share the same L-prime implicates. Indeed:

Definition 3.15 (L-equivalence) *Let Σ, Φ be two formulas from $PROP_{PS}$ and L be a finite subset of L_{PS}. Σ and Φ are said* L-equivalent, *noted $\Sigma \equiv_L \Phi$, iff $forget(\Sigma, Lit(\Sigma) \setminus L) \equiv forget(\Phi, Lit(\Phi) \setminus L)$.*

Example 3.26 *Let $\Sigma = (a \Rightarrow b) \wedge (b \Rightarrow c)$ and $\Phi = (a \Rightarrow d) \wedge (d \Rightarrow c)$. Let $L = \{\neg a, c\}$. Then Σ and Φ are L-equivalent.*

Corollary 3.5 *Let Σ, Φ be two formulas from $PROP_{PS}$ and L be a finite subset of L_{PS}. $\Sigma \equiv_L \Phi$ holds iff $PI_L(\Sigma) = PI_L(\Phi)$.*

As a specific case, if $\Sigma \equiv \Phi$ holds, then $PI_L(\Sigma) = PI_L(\Phi)$. While the corollary above shows that the set of L–prime implicates of a formula Σ does not change when Σ is transformed into an equivalent one, the notion of L–prime implicates still has a syntactic flavour; especially, it is not the case that any clause equivalent to a L-prime implicate also is a L-prime implicate (just consider tautologies).

Clearly enough, a covering property for L-prime implicates results from both Proposition 3.4 and Proposition 3.13. Indeed, for every clause γ, $forget(\Sigma, Lit(\Sigma) \setminus L) \models \gamma$ holds iff there exists π in $PI_L(\Sigma)$ s.t. $\pi \models \gamma$. Since for every clause γ built up from L, $\Sigma \models \gamma$ holds iff $forget(\Sigma, Lit(\Sigma) \setminus L) \models \gamma$ holds, we have $\Sigma \models \gamma$ iff there exists π in $PI_L(\Sigma)$ s.t. $\pi \models \gamma$ for all such clauses. Another consequence is that $PI_L(\Sigma) \equiv forget(\Sigma, Lit(\Sigma) \setminus L)$. Thus, in contrast to the (standard) prime implicates situation, some information is lost (in the general case) when focusing on clauses built up from L. However, Σ and $PI_L(\Sigma)$ are equivalent for answering queries built up from literals of L (and conjunctions of such clauses) in the sense that, for every clause γ s.t. $Lit(\gamma) \subseteq L$, we have $\Sigma \models \gamma$ iff $PI_L(\Sigma) \models \gamma$. Accordingly, $PI_L(\Sigma)$ can be considered as an approximate compilation of Σ, which is complete only for queries restricted to clauses built up from L [del Val, 1995].

Example 3.27 *Let $L = \{\neg a, b\}$ and $\Sigma = (a \Rightarrow b) \wedge (b \Rightarrow c)$. b is not a logical consequence of Σ since it is not entailed by the (unique) L–prime implicate $(\neg a \vee b)$ of Σ.*

Subsequently, every L–prime implicate of Σ is entailed by at least one L–prime implicate of every logically stronger formula. Another important consequence of the fact that $PI_L(\Sigma) \not\equiv \Sigma$ in the general case is the fact that the equivalence between the conjunction of all prime implicates of Σ and the disjunction of all prime implicants of Σ does not hold any longer in the situation where $L \neq L_{PS}$. Actually, the disjunction of all L-prime implicants of Σ is logically stronger but typically not equivalent to Σ.

Example 3.28 *Let $L = \{a\}$ and $\Sigma = (a \vee b)$. a is a L-prime implicant of Σ but Σ has no L-prime implicates.*

It is easy to see that strengthening (or weakening) Σ may heavily change the set of its L–prime implicates. Contrastingly, as a direct consequence of Proposition 3.11, is the following monotonicity property:

Corollary 3.6 *Let Σ be a formula from $PROP_{PS}$ and L, L' be two finite subsets of L_{PS}. If $L \subseteq L'$ holds then $PI_L(\Sigma) \subseteq PI_{L'}(\Sigma)$ holds.*

In particular, this result shows that the number of L-prime implicates of Σ never increases when L is reduced.

Thanks to Proposition 3.11, every algorithm for computing prime implicates can be easily used for generating L–prime implicates. However, such a filtering approach is not always so good since it may lead to the generation of exponentially many clauses (see Section 3.4) though only a few L–prime implicates are to be computed (especially when the number of elements in L is small).

Alternatively, we can take advantage of Proposition 3.13 to make the job, but this requires to compute first $forget(\Sigma, Lit(\Sigma) \setminus L)$. While Definition 3.14 shows us a way to generate this formula, it cannot be used as it is in every situation since the size of the generated formula is exponential in the number of variables to be forgotten in the worst case if no simplification can be done.

There also exist resolution-based techniques for computing $forget(\Sigma, Lit(\Sigma) \setminus L)$. Specifically, *directional resolution* [Dechter and Rish, 1994] can be used to compute a CNF formula equivalent to $forget(\Sigma, Lit(\Sigma) \setminus L)$ from a CNF formula Σ. Let us consider any A-ordering over $Var(\Sigma)$ in which the variables of $Var(L)$ are the last ones. Without loss of generality, let us assume that these variables are $\{x_{k+1}, \ldots, x_n\}$. Every clause of Σ is given a unique index, the rank of its highest variable w.r.t. the A-ordering (the clause is said put in the *bucket* of that variable). Then, all the resolvents that can be generated from clauses from the first bucket (i.e., those clauses containing the first variable x_1 w.r.t. the A-ordering) are computed and put in their corresponding bucket (which is necessarily associated to a highest rank). If literal x_1 (resp. $\neg x_1$) belongs to L, every clause of the first bucket containing x_1 (resp. $\neg x_1$) is given another index, the one it would had whether x_1 (resp. $\neg x_1$) would not belong to it. The algorithm then resumes from the second bucket and processes them up to the k^{th} one. Once the k^{th} bucket has been processed, all the clauses indexed by $k+1$ to n form a CNF formula that is equivalent to $forget(\Sigma, Lit(\Sigma) \setminus L)$ [17]. Clearly enough, the resulting formula can be exponentially larger than the original one Σ and, if the choice of any A-ordering does not matter w.r.t. the correctness of this algorithm (as long as it respects the requirement above), it may have a strong impact on the efficiency of the approach. Interestingly, given an A-ordering, any clause generated by directional resolution has a kernel deduction [del Val, 1999]. The converse does not hold: it is easy to show that directional resolution is not complete in consequence finding. Nevertheless, as the core of the well-known Davis and Putnam elimination procedure for the satisfiability problem SAT [Davis and Putnam, 1960], directional reolution is complete in proof finding. This is not so suprising since a propositional formula Σ is satisfiable iff the closed quantified boolean formula $\exists Var(\Sigma)\Sigma$ is valid, and a propositional equivalent to the latter can be computed through variable elimination. More sophisticated approaches to the computation of $forget(\Sigma, Lit(\Sigma) \setminus L)$ based on variable elimination also exist (see [Kohlas *et al.*, 1999b] [Kohlas *et al.*, 1999a]). They can be directly used for literal forgetting in the case where the set $Lit(\Sigma) \setminus L$ of literals to be forgotten is closed under negation (in this case, literal forgetting reduces to variable forgetting). The purpose of such decomposition techniques is to avoid redundant computations by maintaining the computations as local as possible. They are based on the following properties:

Proposition 3.14 *Let Σ be a formula from $PROP_{PS}$. For every subset $X = \{x_1, \ldots, x_m\}$ of PS, we note $\exists X\Sigma$ the formula $\exists x_1 \ldots \exists x_m \Sigma$. Let X, Y be two subsets of PS.*

- *If $X \subseteq Y$ holds, then $\exists(X \cup Y)\exists X\Sigma \equiv \exists(X \cup Y)\Sigma$.*
- *Let Φ be a formula from $PROP_{PS}$. If $Var(\Sigma) \cap X = \emptyset$ and $Var(\Phi) \cap Y = \emptyset$, then $\exists X(\Sigma \wedge \Phi) \equiv \Sigma \wedge \exists(X \cup Y)\Phi$* [18].

In the case where L is small enough, a way to generate $forget(\Sigma, Lit(\Sigma) \setminus L)$ consists in generating a Shannon tree of Σ around its $Var(\Sigma) \cap Var((L)$ (in any order). In this tree, the paths labelled with literals of L, only, are considered. For every leaf of such a path, the restriction formula of Σ rooted at that leaf is checked for satisfiability; if it is satisfiable, then the conjunction of every literal from L labelling the path leading to this leaf is collected. When the tree has been totally searched, the set of such terms is a DNF of $forget(\Sigma, Lit(\Sigma) \setminus L)$.

[17] Actually, this is an extension of Dechter and Rish's algorithm; the original version is dedicated to variable elimination, not literal forgetting.

[18] The analogous property for literal forgetting does not hold.

There also exist computational techniques that do not require a formula equivalent to $forget(\Sigma, Lit(\Sigma) \setminus L)$ to be generated first. Thus, when the size of L is sufficiently small, an approach to generate $PI_L(\Sigma)$ consists in considering all the clauses built up from L in a systematic way, taking into account the logically strongest ones in priority (a way to organize this search consists in developping a SE-tree (Set-Enumeration tree) [Rymon, 1992] so as to avoid considering twice or more the same clause). Such clauses are successively checked for being implicates of Σ; if so, they are necessarily prime ones and no logically weakest clauses have to be considered (they are not generated within the SE-tree).

Resolution-based approaches to the direct generation of L–prime implicates also exist. Finger and Genesereth [Finger and Genesereth, 1985] introduce *residue resolution*, a general resolution strategy enriched with an additional *skipping rule* (called *make assumption*). The notion of *skipped literal* is central to many resolution strategies used to compute L-prime implicates. To every clause γ that has been derived so far using the strategy under consideration (i.e., a clause that has an admissible resolution proof) and contains a literal l from L, the skipping rule can be applied so as to remember in γ that l belongs to L: l is said skipped within Σ. Typically, such clauses γ are represented as s-clauses, i.e., ordered pairs of sets of literals, the first set containing the skipped literals and the second set the remaining ones. Through the distinction between skipped and unskipped literals within a clause, the purpose is to limitate the set of admissible proofs to be searched, while ensuring a *restricted form of completeness in consequence finding*: for every L-implicate γ of Σ, there must exist a clause γ' that has an admissible resolution proof from Σ and $\gamma' \models \gamma$ holds. In resolution strategies using a skipping rule, this goal is achieved by limiting the application of inference rules each time an unskipped literal is considered. To illustrate this idea, consider the extreme case where $L = \emptyset$: in this situation, there is only one element in $PI_L(\Sigma)$ (the empty clause) if Σ is contradictory, and no element in the remaining case, provided that Σ is not valid (which can be checked efficiently when Σ is in CNF). In such a situation, proof finding is sufficient, and since no literal is to be skipped, any valuable resolution strategy that is complete in the restricted form of consequence finding presented above should degenerate into a resolution strategy for which completeness in proof finding is sufficient (and completeness in consequence finding would be a drawback). Stated otherwise, in any valuable resolution-based strategy for the direct generation of L-prime implicates, the underlying resolution strategy (i.e., the one obtained by ruling out the skipping rule) must not be complete in consequence finding, and the skipping rule must be mandatory to achieve the restricted form of consequence finding [19].

[19] Quite surprisingly, this is not the case for residue resolution; w.r.t. this strategy, only the unskipped literals of a clause can be resolved; since the skipping rule does not have

Among the resolution-based approaches to the generation of L-prime implicates is Bossu and Siegel's saturation algorithm [Bossu and Siegel, 1985] for computing positive implicates (i.e., $L = L^+_{PS}$). This algorithm involving A-ordering actually works in a restricted form of clausal first-order logic. Siegel [Siegel, 1987] also presents a family of production algorithms that are intended to compute prime implicates belonging to a given stable production field. These algorithms are based on SL-resolution [Kowalski and Kuhner, 1971], an efficient version of ordered linear resolution (involving C-ordering). Within such resolution-based algorithms, literals belonging to the production field are skipped. Clearly enough, these algorithms can be used to compute L-prime implicates since the set of all clauses containing only literals from L is a stable production field. Cazalens [Cazalens, 1992] also presents resolution-based techniques for deriving clauses belonging to a production field (provided that it satisfies some additional properties, see [Cazalens, 1992] for details). Demolombe and Fariñas del Cerro [Demolombe and del Cerro, 1991] point out a resolution strategy oriented towards the generation of clauses that contain a given literal; such a strategy can be used to derive the prime implicates of a formula that contains such a literal [20].

Recently, del Val [del Val, 1999] introduces *SFK-resolution*, a restricted form of kernel resolution (SFK stands for "Skip-Filtered Kernel") that is complete in restricted consequence finding (only the fact that all the prime implicates belonging to a given stable production field are derivable is guaranteed). In the following, we present SFK-resolution, focusing on the production field consisting of all the clauses containing only literals from L. Nevertheless, let us stress that SFK-resolution is not specifically dedicated to the computation of L-prime implicates, but can also be used to derive Φ-prime implicates and $\langle L, \Phi\rangle$-prime implicates, among others. SFK-resolution is defined as follows (for simplicity, all the clauses are assumed reduced and represented by sets of literals):

Definition 3.16 ($\mathcal{L}_T$-SFK-resolution proof) *Let Σ be a CNF formula from $PROP_{PS}$ consisting of reduced clauses. Let $\mathcal{L}_T$ be a stable production field. Let γ be a reduced clause from $PROP_{PS}$. Given any A-ordering over $Var(\Sigma)$, an* $\mathcal{L}_T$-SFK-resolution proof *of γ from Σ is a finite sequence γ_1, ..., γ_n of s-clauses from $PROP_{PS}$ s.t.:*

to be applied in priority w.r.t. the resolution one, every resolution proof is a residue resolution proof. Thus, because the underlying resolution strategy for residue resolution is general resolution (that is complete in consequence finding), residue resolution is also complete in consequence finding. Hence, the objective of restricting the set of resolution proofs to be searched is not reached.

[20] It is important to keep in mind that a L-implicate is a clause s.t. *all* the literals in it belongs to L and *not* a clause containing *at least one* literal from L. Nevertheless, using Shannon decomposition, it is not difficult to give a characterization of such prime implicates; indeed, the prime implicates π of a non-valid formula Σ s.t. $\pi \equiv (l \vee \gamma)$ where l is a given literal of L_{PS} and γ is a clause s.t. $Var(\gamma) \cap Var(l) = \emptyset$ are exactly the elements of $PI^{forget(\Sigma,\{l,\neg l\})}(\Sigma_{l \leftarrow false})$.

- γ *is equivalent to* γ_n*,*
- *For every* i *in* $1 \,..\, n$*,* γ_i *is not a valid clause,*
- *For every* i *in* $1 \,..\, n$*, either:*
  - $kernel(\gamma_i) \in \Sigma$ *and* $skip(\gamma_i) = \emptyset$ *(input clause); or*
  - γ_i *is the resolvent of two clauses* γ_j *and* γ_k *with* $j < i$ *and* $k < i$*. In this case, the literals resolved upon are respectively in* $kernel(\gamma_j)$ *and* $kernel(\gamma_k)$*,* $kernel(\gamma_i)$ *is the set of all literals of* γ_i *that are larger or equal to the literals resolved upon w.r.t. the given A-ordering, and* $skip(\gamma_i)$ *is the set of (strictly) smaller literals. Finally,* $skip(\gamma_i) \subseteq \mathcal{L}_T$ *(the resolvent is said* $\mathcal{L}_T$-acceptable*).*

In this section, we are specifically interested in L-SFK-resolution proofs, corresponding to the case where $\mathcal{L}_T$ is the set of all clauses containing only literals from L.

Example 3.29 *Let* $\Sigma = (a \vee b) \wedge (a \vee c \vee d) \wedge (\neg b \vee \neg c)$*. Let us consider the A-ordering s.t.* $a < b < c < d$*. If* $\{a\} \subseteq L$*, the following sequence is a L-SFK-resolution proof of* $a \vee d$ *from* Σ*:*

1. $\langle \emptyset, \{a, b\} \rangle$ *(input clause)*
2. $\langle \emptyset, \{\neg b, \neg c\} \rangle$ *(input clause)*
3. $\langle \{a\}, \{\neg c\} \rangle$ *(resolution between 1. and 2. upon b)*
4. $\langle \emptyset, \{a, c, d\} \rangle$ *(input clause)*
5. $\langle \{a\}, \{d\} \rangle$ *(resolution between 3. and 4. upon c)*

Accordingly, SFK-resolution enables deriving the $\{a, d\}$*-prime implicate* $(a \vee d)$ *of* Σ*. Note that, despite the fact that* $(a \vee d)$ *is an implicate of* Σ*, the resolution proof above would not be a L-SFK-resolution proof any longer would L not contain a.*

Proposition 3.15 *SFK-resolution is* complete in characteristic consequence finding *in the sense that for every implicate* γ *of a CNF formula* Σ *s.t.* γ *belongs to a (recursive) production field* $\mathcal{L}_T$*, there exists an implicate of* Σ *belonging to* $\mathcal{L}_T$ *that is logically stronger than* γ *and has a* $\mathcal{L}_T$*-SFK-resolution proof from* Σ*.*

del Val [del Val, 1999] presents two search strategies for the exhaustive exploration of the set of SFK-resolution proofs. The first one is based on *bucket elimination* and it generalizes the methods of [Dechter and Rish, 1994], while the second one is based on *incremental saturation* and is specially suited to the case where the input formula is not initially completely specified.

In the bucket elimination strategy, every clause γ of the input CNF formula Σ is associated to a subset of $Var(\gamma)$, through an indexing function $I_{\mathcal{L}_T}$ (when $x_i \in I_{\mathcal{L}_T}(\gamma)$, γ is said put in the bucket i). The indexing function $I_{\mathcal{L}_T}$ must be *acceptable for* $\mathcal{L}_T$, i.e., for every clause γ of Σ, for every variable $x_i \in Var(\gamma)$, $x_i \in I_{\mathcal{L}_T}(\gamma)$ iff γ can be resolved upon x_i with a clause of Σ and the corresponding resolvent is $\mathcal{L}_T$-acceptable (its skip is a clause from $\mathcal{L}_T$). Several choices are possible for an indexing function, especially the one that consists in associating to every γ sorted in ascending order w.r.t. the chosen A-ordering the kernel variables of its largest prefix $\{l_1, \ldots l_{k+1}\}$ s.t. every l_i (i in 1 .. k) belongs to L. Once every clause of Σ has been indexed, bucket elimination processes the buckets in ascending order w.r.t. the A-ordering. At step i, it computes all the resolvents upon the i^{th} variable x_i (w.r.t. the A-ordering) of all the clauses γ of Σ s.t. $x_i \in I_{\mathcal{L}_T}(\gamma)$. Among them, only the $\mathcal{L}_T$-acceptable resolvents are considered and they are added to their corresponding buckets by calling the indexing function $I_{\mathcal{L}_T}$. As shown in [del Val, 1999], this strategy is a terminating and exhaustive search strategy for $\mathcal{L}_T$-SFK-resolution. Accordingly, the $\mathcal{L}_T$-prime implicates of Σ are among the generated clauses.

It is interesting to note that in the situation where $\mathcal{L}_T = L_{PS}$, every possible resolvent is $\mathcal{L}_T$-acceptable and the indexing function that associates to every clause its kernel is acceptable for $\mathcal{L}_T$ as well. Clearly enough, in this restricted case, bucket elimination corresponds exactly to Tison's method for computing prime implicates. Contrariwise, in the situation where $\mathcal{L}_T = \emptyset$, every clause can be indexed only by its smallest variable w.r.t. the A-ordering (resolving on any other kernel variable cannot yield any $\mathcal{L}_T$-acceptable resolvent). Here, bucket elimination is identical to directional resolution [Dechter and Rish, 1994].

Another terminating and exhaustive search strategy for $\mathcal{L}_T$-SFK resolution is incremental saturation. In this strategy, a set of input clauses γ is processed in turn until no unsubsumed $\mathcal{L}_T$-acceptable resolvents can be generated from the available clauses (the corresponding set is said $\mathcal{L}_T$-saturated). Given an A-ordering, the expansion of a set of input clauses γ into a current $\mathcal{L}_T$-saturated set of clauses Σ requires to maintain two distinct sets of buckets: one passive bucket for the clauses of Σ and an active one for indexing the clauses γ and all the newly generated clauses. Bucket elimination is performed on the active set of buckets in ascending order w.r.t. the A-ordering: at each step i, all the resolvents upon x_i of all the clauses that are indexed by x_i in both the passive and active sets of buckets are considered, and the $\mathcal{L}_T$-acceptable resolvents are added to their corresponding buckets in the active set by calling the indexing function $I_{\mathcal{L}_T}$. When finished, the buckets of the active set are emptied into the corresponding buckets of the passive set to prepare the mechanism for receiving new input clauses. Clearly enough, the use of two sets of buckets enables never resolving two clauses of the current $\mathcal{L}_T$-saturated set of clauses (accordingly, incremental saturation can be viewed as an extension of the algorithm by Kean and Tsiknis [Kean and Tsiknis, 1990b]).

Developping L–prime implicates algorithms that are based on the structure of the input formula Σ is not easy, even in restricted cases. Especially, Proposition 3.7 cannot be adaptated to the L–prime implicates situation since the L–prime implicates of $(\Phi \wedge \Sigma)$ when Σ is a literal cannot be recovered from $PI_L(\Phi)$ and Σ in the general case (as mentioned previously, there is no decomposition property for literal forgetting within a conjunctive formula).

Example 3.30 *Let* $\Phi = (a \vee b)$, $\Sigma = \neg b$ *and* $L = \{a\}$. *While there is no* L*–prime implicate of* Φ, a *is a* L*–prime implicate of* $\Phi \wedge \Sigma$.

Things looks better whenever Σ is given as a disjunctive formula. Indeed, we can take advantage of the decomposition property $forget(\Sigma_1 \vee \Sigma_2, L) \equiv (forget(\Sigma_1, L) \vee forget(\Sigma_2, L))$. As a consequence of Propositions 3.13 and 3.9, we have the following distribution property for L-prime implicates:

Proposition 3.16 (distribution) *Let* $\Sigma_1, \ldots, \Sigma_k$ *be* k *formulas from* $PROP_{PS}$ *and* L *be a finite subset of* L_{PS}. $PI_L(\Sigma_1 \vee ... \vee \Sigma_k) = min(\{(\pi_{\Sigma_1} \vee ... \vee \pi_{\Sigma_k}); \pi_{\Sigma_1} \in PI_L(\Sigma_1), ..., \pi_{\Sigma_k} \in PI_L(\Sigma_k)\}, \models)$.

Example 3.31 *Let* $\Sigma_1 = (a \vee b) \wedge c$, $\Sigma_2 = (\neg a \vee b) \wedge (b \vee c)$ *and* $L = \{b, c\}$. $PI_L(\Sigma_1) = \{c\}$ *and* $PI_L(\Sigma_2) = \{(b \vee c)\}$. *Accordingly,* $PI_L(\Sigma_1 \vee \Sigma_2) = \{(b \vee c)\}$ *(up to logical equivalence).*

Algorithms for computing L-prime implicates can benefit from this proposition. Corollary 3.3 as well as the various heuristics mentioned in the previous section can still be used here to reach an efficient implementation of the distribution property. Additionally, Corollary 3.4 can also be adapted to the case of L-prime implicates. Indeed, as evoked above, a DNF of $forget(\Sigma, Lit(\Sigma) \setminus L)$ can be easily generated from a DNF of Σ by removing in every satisfiable term all the literals from $Lit(\Sigma) \setminus L$. Once such a DNF have been generated, Proposition 3.13 shows how L–prime implicates can be generated taking advantage of Corollary 3.4. Thus, the algorithm by Castell [Castell, 1997] for computing prime implicates has been adapted to the case of L–prime implicates: instead of computing a DNF of Σ in a first step, a DNF of $forget(\Sigma, Lit(\Sigma) \setminus L)$ is computed by forgetting useless literals. In the same vein, the algorithm by Madre and Coudert [Madre and Coudert, 1991] has been extended to the computation of L-prime implicates (let us recall that the first step of this algorithm is the computation of a BDD of the input formula, that is just a compact representation of a DNF formula).

Finally, Proposition 3.8 can be extended to the L-prime implicates situation: any algorithm for computing $\langle L, \Phi\rangle$-prime implicates can also be used to derive L-prime implicates.

Proposition 3.17 *Let Φ and Σ be formulas from $PROP_{PS}$. Let L be any finite subset of L_{PS}. We have $PI_L(\Phi \wedge \Sigma) = min(PI_L(\Phi) \cup PI_L^{\Phi}(\Sigma), \models)$ (up to logical equivalence).*

Example 3.32 *Let $\Phi = (a \Rightarrow b) \wedge (b \vee c) \wedge (a \vee d) \wedge (b \Rightarrow e)$, $\Sigma = a$ and $L = \{a, b, c\}$. $PI_L(\Phi \wedge \Sigma) = min(PI_L(\Phi) \cup PI_L^{\Phi}(\Sigma), \models) = min(\{(b \vee c)\} \cup \{a, b\}, \models) = \{a, b\}$.*

Accordingly, it is possible to take advantage of algorithms for deriving $\langle L, \Phi\rangle$-prime implicates (like those based on SOL-resolution, cf. Section 3.3.4) to compute L-implicates.

3.3.3 Φ–prime implicates

Let us now turn to Φ–prime implicates. Clearly enough, the Φ–prime implicates of Σ are its logically strongest Φ–implicates, i.e., its prime Φ–implicates. Indeed, the set of clauses that are not implied by Φ is a stable production field. Accordingly, the Φ–prime implicates of Σ are selected prime implicates. Since every Φ–implicate of Σ is an implicate of $\Phi \wedge \Sigma$, the Φ–prime implicates of Σ are among the prime implicates of $\Phi \wedge \Sigma$. More precisely, they are the prime implicates of $\Phi \wedge \Sigma$ that are not entailed by Φ.

Proposition 3.18 *Let Φ and Σ be two formulas from $PROP_{PS}$. $PI^{\Phi}(\Sigma) = \{\pi \in PI(\Sigma \wedge \Phi); (\forall \pi_{\Phi} \in PI(\Phi))(\pi \not\equiv \pi_{\Phi})\}$.*

Example 3.33 *Let $\Phi = (a \Rightarrow b)$ and $\Sigma = (b \Rightarrow c)$. Then $PI(\Phi \wedge \Sigma) = \{(\neg a \vee b), (\neg b \vee c), (\neg a \vee c)\}$ and $PI(\Phi) = \{(\neg a \vee b)\}$. Accordingly, $PI^{\Phi}(\Sigma) = \{(\neg b \vee c), (\neg a \vee c)\}$ (up to logical equivalence).*

Thus, the Φ–prime implicates of Σ are just those prime implicates of $\Phi \wedge \Sigma$ that are not prime implicates of Φ (up to logical equivalence). As a consequence, the set of Φ–prime implicates of Σ always is finite. In contrast to $PI(\Sigma)$, $PI^{\Phi}(\Sigma)$ can easily be empty (this is exactly the case each time $\Phi \models \Sigma$ holds).

The concept of Φ–prime implicate can be viewed as a strict generalization of the concept of prime implicate. Indeed, prime implicates can always be viewed (and computed) as Φ–prime implicates (except in the degenerated case where Σ is valid); especially, we have $PI(\Sigma) = PI^{true}(\Sigma)$ whenever Σ is not valid. However, the converse does not hold. Thus, in contrast to L–prime implicates, the Φ–prime implicates of Σ cannot be characterized as the prime implicates of some formula in the general case.

Example 3.34 *Let $\Phi = (a \Rightarrow c)$; we have $PI^{\Phi}(\Sigma) = \{(\neg a \vee b), (\neg b \vee c)\}$ and $\Sigma = (a \Rightarrow b) \wedge (b \Rightarrow c)$. No formula admits this set of clauses as prime implicates since if $(\neg a \vee b)$ and $(\neg b \vee c)$ were prime implicates of it, it should necessarily be the case for $(\neg a \vee c)$ as well.*

Clearly enough, Proposition 3.18 gives a way to compute Φ–prime implicates, taking advantage of algorithms for computing prime implicates: it is sufficient to select those prime implicates of $\Phi \wedge \Sigma$ that are not prime implicates of Φ.

For this purpose, the *incremental algorithms* have been proposed so far for computing prime implicates, e.g., [Kean and Tsiknis, 1990b] [Jackson, 1992] [de Kleer, 1992] [Ngair, 1993] [del Val, 1999], are of the utmost value; as shown in Section 3.3.1, such algorithms aim at computing the prime implicates of $\Phi \wedge \Sigma$ where Φ is given by its prime implicates, without computing everything from the scratch (generating implicates of Φ is avoided). While the complexity of computing prime implicates incrementally is not lower than the complexity of their non-incremental computation in the worst case (see Section 3.4), some benefits can be achieved in various practical situations. Especially, the (prime) implicates of $\Phi \wedge \Sigma$ that are implicates of Φ can be efficiently detected once $PI(\Phi)$ has been computed (i.e., they can be detected in $\mathcal{O}(|PI(\Sigma)|)$ time) due to the covering property satisfied by prime implicates, hence they can be removed as soon as they are generated.

More efficient techniques aim at avoiding the generation of clauses that are consequences of Φ alone. Finger [Finger, 1987] shows how set-of-support resolution can be used so as to generate clauses that are logical consequences of a set of clauses Φ and a newly added CNF formula as set of support. Thus, no resolution proof involving clauses from Φ alone is to be considered. As we shall see in the next section, ordered linear resolution can also be exploited in a similar way so as to perform direct search for the clauses that are not consequences of Φ. Finally, let us stress that algorithms for computing prime implicates from a given stable production field (like SFK-resolution [del Val, 1999] and SOL-resolution [Inoue, 1992]) can also be used to derive Φ-prime implicates (since the set of all clauses that are not entailed by Φ is a stable production field).

Φ–prime implicates exhibit only few interesting metatheoretic properties. Since both Φ and Σ are considered up to logical equivalence in Definition 3.6, we obviously have that if $\Phi \equiv \Phi'$ and $\Sigma \equiv \Sigma'$ hold, then $PI^{\Phi}(\Sigma) = PI^{\Phi'}(\Sigma')$. Strengthening or weakening Φ (resp. Σ) may deeply change the set $PI^{\Phi}(\Sigma)$. However, since Φ–prime implicates are prime Φ–implicates and $PI^{\Phi}(\Sigma)$ always is a finite set, a covering property exists for Φ–implicates: a clause γ is a Φ–implicate of Σ iff there exists a Φ–prime implicate π of Σ s.t. $\pi \models \gamma$. As a consequence, the Φ–prime implicates of a formula can be related to the Φ–prime implicates of any logically stronger formula in the following way: if $\Sigma' \models \Sigma$ holds, then for every Φ–prime implicate π of Σ, there exists a Φ–prime implicate π' of Σ' s.t. $\pi' \models \pi$ holds. Contrastingly, the Φ–prime implicates of Σ cannot be related to the Φ'–prime implicates of Σ, where Φ' is any formula logically stronger than Φ; indeed, strengthening Φ generally leads to generate some new implicates but also to remove some existing ones.

Example 3.35 *Let $\Phi = (a \Rightarrow b)$ and $\Phi' = (a \Rightarrow b) \wedge b$. Let $\Sigma = a$. We have $PI^{\Phi}(\Sigma) = \{a, b\}$ and $PI^{\Phi'}(\Sigma) = \{a\}$. While $\Phi' \models \Phi$ holds, b is not a logical consequence of a.*

Additionally, Σ is Φ-equivalent to the conjunction of its Φ–prime implicates; thus, no information is lost by replacing Σ by its Φ–prime implicates given Φ.

Example 3.36 *Let $\Phi = (a \Rightarrow b)$ and $\Sigma = a$. We have $PI^{\Phi}(\Sigma) = \{a, b\}$ and $\Sigma \equiv_{\Phi} PI^{\Phi}(\Sigma)$ holds.*

As a consequence, if $\Sigma \equiv_{\Phi} \Sigma'$, then $PI^{\Phi}(\Sigma) = PI^{\Phi}(\Sigma')$: Φ-equivalent formulas have the same set of Φ–prime implicates. However, the Φ-prime implicates of Σ cannot be reduced considering Φ-equivalence. Indeed, if π is a Φ-prime implicate of Σ, then every clause π' s.t. $\pi' \equiv_{\Phi} \pi$ needs not be a Φ–prime implicate of Σ. Moreover, Φ-equivalent Φ-prime implicates of Σ are not identified within $PI^{\Phi}(\Sigma)$ in the general case.

Example 3.37 *Let $\Phi = (a \Leftrightarrow b)$ and $\Sigma = (a \Rightarrow c)$. $(\neg a \vee c)$ and $(\neg b \vee c)$ are Φ–equivalent clauses but are distinct Φ–prime implicates of Σ nevertheless. Additionally, $(\neg a \vee \neg b \vee c)$ is Φ–equivalent to both clauses but is not a Φ–prime implicate of Σ.*

In the general case, the Φ–prime implicates of Σ cannot be characterized through the prime implicates of Φ and those of Σ in a computationally helpful way, except by taking advantage of Proposition 3.18. Nevertheless, the Φ–prime implicates of a conjunctive formula $\Sigma = \Sigma_1 \wedge ... \wedge \Sigma_n$ can be computed from prime implicates of its conjuncts Σ_i $(i \in 1 \,..\, n)$. Indeed:

Proposition 3.19 *Let Φ be a formula from $PROP_{PS}$ and Σ_1, ..., Σ_n be formulas from L_{PS}. Let Φ_1 be Φ and Φ_{i+1} be $\Phi_i \wedge \Sigma_i$ for $i \in 1 \,..\, n-1$. We have $PI^{\Phi}(\Sigma_1 \wedge ... \wedge \Sigma_n) = min(PI^{\Phi_1}(\Sigma_1) \cup ... \cup PI^{\Phi_n}(\Sigma_n), \models)$ (up to logical equivalence).*

Example 3.38 *Let $\Phi = ((a \wedge b) \Rightarrow c) \wedge (a \Rightarrow d)$ and $\Sigma = (b \wedge \neg c)$. We have $PI^{\Phi}(\Sigma) = min(PI^{\Phi}(b) \cup PI^{(\Phi \wedge b)}(\neg c), \models) = min(\{(\neg a \vee c), b\} \cup \{\neg a, \neg c\}, \models) = \{b, \neg c, \neg a\}$ (up to logical equivalence).*

Another specific case of interest is reached when Σ is a literal. In this situation, Proposition 3.7 can be specialized so as to characterizing the Φ–prime implicates of Σ through the prime implicates of Φ and Σ.

Proposition 3.20 *Let Φ be a formula from $PROP_{PS}$ and Σ be a literal from L_{PS}. If $\Phi \models \Sigma$ holds then $PI^{\Phi}(\Sigma) = \emptyset$; otherwise, if $\Phi \models \neg\Sigma$ holds then $PI^{\Phi}(\Sigma) = \{false\}$; otherwise, $PI^{\Phi}(\Sigma) = \{\Sigma\} \cup \{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\}$.*

Example 3.39 *Let $\Phi = ((a \wedge b) \Rightarrow c) \wedge (a \Rightarrow d)$ and $\Sigma = b$. We have $PI(\Phi) = \{(\neg a \vee \neg b \vee c), (\neg a \vee d)\}$. Accordingly, $PI^{\Phi}(\Sigma) = \{b, (\neg a \vee c)\}$.*

Since $PI(\Phi)$ can be viewed as a compiled form of Φ from which clause deduction is tractable, checking whether $\Phi \models \neg\Sigma$ (resp. $\Phi \models \Sigma$) can be done in time polynomial in the size of $PI(\Phi)$. Hence, $PI^{\Phi}(\Sigma)$ can be computed in time polynomial in the size of $PI(\Phi)$ when Σ is a literal. Interestingly, no deduction checks are to be performed (compare with Proposition 3.7), hence the incremental computation of $PI^{\Phi}(\Sigma)$ from $PI(\Phi)$ and a literal Σ is particularly efficient. However, Proposition 3.20 cannot be extended in a trivial way to the case where Σ is a term since it can be the case that $PI^{\Phi}(\Sigma)$ does not correspond to the set of prime implicates of some formula. Nevertheless, we have:

Proposition 3.21 *Let Φ be a formula from $PROP_{PS}$ and Σ be a term from $PROP_{PS}$. If $\Phi \models \Sigma$ holds then $PI^{\Phi}(\Sigma) = \emptyset$; otherwise, if $\Phi \models \neg\Sigma$ holds then $PI^{\Phi}(\Sigma) = \{false\}$. Otherwise, let us assume without loss of generality that Σ is a reduced DNF, i.e., it does not contain any occurrence of true, of $false$, and no redundant occurrence of any literal. In this situation, the clause $\neg\Sigma$ can be considered as the (necessarily non empty) set of the literals occurring in it. Then $PI^{\Phi}(\Sigma) = min(\Sigma \cup \{\pi$; there exists a (disjunctively interpreted) non-empty subset S_π of $\neg\Sigma$ s.t. $\pi \vee S_\pi \in PI(\Phi)$ and no literal of π belongs to $S_\pi\}, \models)$.*

Example 3.40 *Let $\Phi = ((a \wedge b) \Rightarrow c) \wedge (a \Rightarrow d)$ and $\Sigma = (b \wedge \neg c)$. $PI(\Phi) = \{(\neg a \vee \neg b \vee c), (\neg a \vee d)\}$. Accordingly, $PI^{\Phi}(\Sigma) = \{b, \neg c, \neg a\}$ (up to logical equivalence).*

Clearly, Proposition 3.21 shows how $PI^{\Phi}(\Sigma)$ can be computed in time polynomial in $|PI(\Phi)| + |\Sigma|$ whenever Σ is a term.

We can also derive a distribution property for Φ–prime implicates, from Proposition 3.18 and Proposition 3.9:

Proposition 3.22 (distribution) *Let $\Sigma_1, \ldots, \Sigma_k$ be k formulas from $PROP_{PS}$ and Φ be a formula from $PROP_{PS}$. $PI^{\Phi}(\Sigma_1 \vee ... \vee \Sigma_k) = min(\{(\pi_{\Sigma_1} \vee ... \vee \pi_{\Sigma_k}); \pi_{\Sigma_1} \in PI^{\Phi}(\Sigma_1), ..., \pi_{\Sigma_k} \in PI^{\Phi}(\Sigma_k)$ and $\Phi \not\models (\pi_{\Sigma_1} \vee ... \vee \pi_{\Sigma_k})\}, \models)$.*

Example 3.41 *Let $\Phi = (a \Rightarrow b)$, $\Sigma_1 = a$ and $\Sigma_2 = c$. We have $PI^{\Phi}(\Sigma_1 \vee \Sigma_2) = \{(a \vee c), (b \vee c)\}$, $PI^{\Phi}(\Sigma_1) = \{a, b\}$ and $PI^{\Phi}(\Sigma_2) = \{c\}$.*

Clearly enough, the various heuristics for avoiding generating non-minimal clauses for the prime implicates version of the distribution property can also be used in the Φ–prime implicates case. However, as in the standard prime implicates situation, implementing the distribution property above does not give rise to an algorithm that runs in time polynomial in the input size in the general case (even when Φ is given by the set of its prime implicates).

Combining this distribution property with Proposition 3.21 gives an algorithm for computing the Φ–prime implicates of every Σ given in DNF (just like Proposition 3.4 shows how the prime implicates of a DNF formula can be computed), once Φ has been turned into the set of its prime implicates.

3.3.4 $\langle L, \Phi \rangle$–prime implicates

Let us finally investigate the metatheoretic properties of $\langle L, \Phi \rangle$–prime implicates. Since the concept of $\langle L, \Phi \rangle$–prime implicate is more general than the concept of L–prime implicate (just set Φ to $true$ to recover the latter from the former) when Σ is not a valid formula, and also more general than the concept of Φ–prime implicate (just set L to L_{PS} to recover the latter from the former), $\langle L, \Phi \rangle$–prime implicates cannot exhibit properties that L–prime implicates or Φ–prime implicates do not satisfy in the general case.

Conversely, many of the properties of Φ–prime implicates listed above are satified by $\langle L, \Phi \rangle$–prime implicates. Indeed, a $\langle L, \Phi \rangle$–prime implicate of Σ is just a Φ–prime implicate of Σ containing literals from L, only. This suggests a way to derive the $\langle L, \Phi \rangle$–prime implicates of Σ from the Φ–prime implicates of Σ, using a filtering process similar to the one given by Proposition 3.11.

Moreover, every $\langle L, \Phi \rangle$–implicate π of Σ is a L–implicate of $\Phi \wedge \Sigma$ that is not a L–implicate of Φ. As a consequence:

Proposition 3.23 *Let Φ and Σ be two formulas from $PROP_{PS}$ and L be a finite subset of L_{PS}. $PI_L^{\Phi}(\Sigma) = \{\pi \in PI_L(\Sigma \wedge \Phi); (\forall \pi_\Phi \in PI_L(\Phi))(\pi \not\equiv \pi_\Phi)\}$.*

Example 3.42 *Let $L = \{\neg a, c, d\}$, $\Phi = (a \Rightarrow b) \wedge d$ and $\Sigma = (b \Rightarrow c)$. $PI_L(\Sigma \wedge \Phi) = \{(\neg a \vee c), d\}$. $PI_L(\Phi) = \{d\}$. $PI_P^{\Phi}(\Sigma) = \{(\neg a \vee c)\}$ (up to logical equivalence).*

Subsequently, $\langle L, \Phi \rangle$–prime implicates can be characterized as Φ–prime implicates. More precisely, we have $PI_L^{\Phi}(\Sigma) = PI^{forget(\Phi, Lit(\Phi) \setminus L)}(forget(\Phi \wedge \Sigma, Lit(\Phi \wedge \Sigma) \setminus L))$.

A covering property for $\langle L, \Phi \rangle$–prime implicates can also be derived since $\langle L, \Phi \rangle$–prime implicates are prime $\langle L, \Phi \rangle$–implicates. Thus, a clause γ is a $\langle L, \Phi \rangle$–implicate of Σ iff there exists a $\langle L, \Phi \rangle$–prime implicate π_Φ of Σ s.t. $\pi_\Phi \models \gamma$ holds.

It is also easy to show that $PI_L^{\Phi}(\Sigma)$ is Φ-equivalent to $PI_L(\Phi \wedge \Sigma)$. Since $PI_L^{\Phi}(\Sigma) = \emptyset$ holds iff $\Phi \models PI_L^{\Phi}(\Sigma)$, we also have $PI_L^{\Phi}(\Sigma) = \emptyset$ iff

$\Phi \models PI_L(\Phi \wedge \Sigma)$. This equivalence will prove helpful in the following (see Section 4.4.2).

More importantly, Proposition 3.23 shows how algorithms for computing L-prime implicates can be used for generating $\langle L, \Phi \rangle$-prime implicates. Of specific interest for this purpose are those incremental L-prime implicate algorithms (like incremental saturation [del Val, 1999], see Section 3.3.2) since they prevent from recomputing $PI_L(\Phi)$ anew when Σ is considered.

Proposition 3.19 can also be generalized to the $\langle L, \Phi \rangle$-prime implicates situation, pointing out a way to compute such prime implicates when Σ is a conjunctive formula (especially, a CNF formula).

Proposition 3.24 *Let Φ be a formula from $PROP_{PS}$ and Σ_1, ..., Σ_n be formulas from L_{PS}. Let L be a finite subset of L_{PS}. Let Φ_1 be Φ and Φ_{i+1} be $\Phi_i \wedge \Sigma_i$ for $i \in 1$ .. $n-1$. We have $PI_L^{\Phi}(\Sigma_1 \wedge ... \wedge \Sigma_n) = min(PI_L^{\Phi_1}(\Sigma_1) \cup ... \cup PI_L^{\Phi_n}(\Sigma_n), \models)$ (up to logical equivalence).*

Example 3.43 *Let $\Phi = ((a \wedge b) \Rightarrow c) \wedge (a \Rightarrow d)$, $\Sigma = (b \wedge \neg c)$ and $L = \{a, b\}$. $PI_L^{\Phi}(\Sigma) = min(PI_L^{\Phi}(b) \cup PI_L^{(\Phi \wedge b)}(\neg c), \models) = min(\{b\} \cup \emptyset, \models) = \{b\}$ (up to logical equivalence).*

Since the set of all clauses that are not entailed by a formula Φ but are built up from a set L of literals is a stable production field, algorithms for computing prime implicates from a given stable production field (like SFK-resolution [del Val, 1999]) can also be used to generate $\langle L, \Phi \rangle$-prime implicates.

Interestingly, sophisticated resolution-based techniques have been pointed out for the direct generation of *new characteristic clauses*, i.e., clauses from a given stable production field that are entailed by a knowledge base when expanded with a new input formula but are not entailed by the knowledge base before expansion. $\langle L, \Phi \rangle$-prime implicates of Σ are just such new characteristic clauses in the case where Φ is the knowledge base, Σ is the input formula and the production field is the set of all clauses built up from the set L of literals.

Extending Siegel's previous results [Siegel, 1987], Inoue [Inoue, 1990] [Inoue, 1991] [Inoue, 1992] presents an approach to directly search for such implicates. His approach relies on *SOL-resolution*, i.e., linear resolution involving C-ordering plus an additional inference rule called *skipping*: SOL stands for "Skipping Ordered Linear". It is based on the fact that efficient refinements of linear resolution, most notably model elimination [Loveland, 1978] and SL-resolution [Kowalski and Kuhner, 1971], can be augmented with a skipping rule so as to render them complete in new characteristic consequence finding, i.e., every implicate of $\Phi \wedge \Sigma$ that belongs to a given stable production field $\mathcal{L}_T$ and is not an implicate of Φ is a logical consequence of a clause that is derivable. The interest of linear resolution strategies in deriving new characteristic clauses can be explained by the fact that in linear resolution, a newly added single clause γ is treated as the top clause of a resolution proof; thus, every resolvent in a linear resolution proof has such a top clause as an ancestor. Considering a clause γ from the CNF formula Σ as a top clause and the clauses from Φ as side clauses is a way to promote the generation of Φ-implicates of Σ. The possibility to take into account a C-ordering is also important for efficiency reasons: while there are $n!$ different linear refutations from the CNF formula) $\neg l_1 \wedge ... \wedge \neg l_n$ and the top clause $(l_1 \vee ... \vee l_n)$, there is only one linear resolution with C-ordering [21].

In order to present SOL-resolution in formal terms, we first need some definitions. Thus, an *extended clause* built up from PS is a list of literals from $L_{PS} \cup \{[l]; l \in L_{PS}\}$. $\{[l]; l \in L_{PS}\}$ is a set of so-called *framed literals*. Framing a literal within a clause is a way to mark it so as to remember that it has been resolved upon previously (i.e., if a clause contains the framed literal $[l]$, then it has an ancestor clause containing l). Thus, an extended clause is just an ordered clause (the rank of every literal is important) containing literals and framed literals. From a logical point of view, an extended clause is independent from its framed literals: outside the resolution machinery, an extended clause is considered as its subclause consisting of the unframed literals, only. Conversely, every clause from $PROP_{PS}$ can be considered as an extended clause (just by considering the literals occurring in it in any fixed order). The purpose of framing is to reduce the search space: within model elimination, resolving a center clause with clauses from Φ that have a literal equal to a framed literal at the right of the literal resolved upon is useless: doing it may only produce clauses that are implied by some previous center clauses [22]. Within SOL-resolution, we consider some specific s-clauses, called *structured clauses*: a structured clause built up from PS is just an ordered pair $\langle \delta, \epsilon \rangle$ consisting of a clause δ from $PROP_{PS}$ and an extended clause ϵ built up from PS. We are now ready to define what a SOL-resolution proof is:

Definition 3.17 ($\mathcal{L}_T$-SOL-resolution proof) *Let Φ be a CNF formula from $PROP_{PS}$ consisting of reduced clauses and Σ be a reduced clause from $PROP_{PS}$. Let $\mathcal{L}_T$ be a stable production field. Let γ be a reduced clause. An $\mathcal{L}_T$-*SOL-resolution proof *of γ from Φ and Σ is a sequence γ_1, ..., γ_n of structured clauses s.t. $\gamma_1 = \langle \emptyset, \Sigma \rangle$, $\gamma_n = \langle \gamma, \emptyset \rangle$, and every center clause $\gamma_{i+1} = \langle \delta_{i+1}, \epsilon_{i+1} \rangle$ ($i \in 1$ .. $n-1$) is generated from the previous center clause γ_i according to the following rules:*

[21] This problem is typically addressed in ordered resolution strategies (e.g., in directional resolution [Dechter and Rish, 1994]; it is also addressed in semantic resolution strategies, through the notion of clash, see [Chang and Lee, 1973].

[22] Shostak [Shostak, 1976] also shows how framed literals can be exploited in a more aggressive way to reduce the search space.

- *δ_{i+1} and ϵ_{i+1} are obtained by applying the following rules:*
  1. *Let $\epsilon_i = \langle l, l_2, ..., l_m \rangle$. δ_{i+1} and ζ_{i+1} are obtained by applying one of the following rules:*
     - **Skip***: If $\delta_i \cup \{l\}$ belongs to $\mathcal{L}_T$, then $\delta_{i+1} = \delta_i \cup \{l\}$ and ζ_{i+1} is obtained by removing l from ϵ_i;*
     - **Resolve***: If there is a side clause $\langle l'_1, ..., l'_r \rangle$ from $\Phi \cup \{\Sigma\}$ s.t. $\neg l = l'_t$, then $\delta_{i+1} = \delta_i$ and $\zeta_{i+1} = \langle l'_1, ..., l'_{t-1}, l'_{t+1}, ..., l'_r, [l], l_2, ..., l_m \rangle$;*
     - **Reduce** *If ϵ_i contains the framed literal $[\neg l]$ (ancestry) or δ_i or ϵ_i contains the unframed literal l twice (merging), then $\delta_{i+1} = \delta_i$ and ζ_{i+1} is obtained by removing the leftmost occurrence of l from ϵ_i.*
  2. *ϵ_{i+1} is obtained from ζ_{i+1} by deleting every framed literal not preceded by an unframed literal in the remainder (truncation).*

Moreover, for every $\gamma_i = \langle \delta_i, \epsilon_i \rangle$ ($i \in 1$ .. n), we have:

- *$\delta_i \vee \epsilon_i$ is not a valid clause,*
- *ϵ_i is not a logical consequence of a previous structured clause ϵ_j (with $j < i$).*

The strategy of deleting implied clauses can be partially implemented by restricting the application of the **Resolve** rule: when ϵ_i can be resolved against an ordered clause of Σ that contains a literal appearing as a framed literal in ϵ_i at the right of the literal resolved upon, then this resolution can be avoided as the corresponding resolvent would be entailed by some previous center clause in the proof.

In this section, we are specifically interested in $\langle L, \Phi \rangle$-SOL-resolution proofs, i.e., SOL-resolution proofs oriented towards the production of $\langle L, \Phi \rangle$-implicates (such implicates are new ones when Φ is considered as the underlying knowledge base and the production field contains all the clauses built up from the set L of literals).

Here is an example of $\langle L, \Phi \rangle$-SOL-resolution proof:

Example 3.44 *Let $\Phi = (a \vee b) \wedge (a \vee c)$ and $\Sigma = (\neg b \vee \neg c)$. Let $L = \{a\}$. The following sequence is a $\langle L, \Phi \rangle$-SOL-resolution proof of a from Φ and top clause Σ:*

1. *$\langle \emptyset, \langle \neg b, \neg c \rangle \rangle$ (top clause)*
2. *$\langle \emptyset, \langle a, [\neg b], \neg c \rangle \rangle$ (resolution with the input clause $a \vee b$)*
3. *$\langle a, \langle \neg c \rangle \rangle$ (skipping of a, then truncation of $[\neg b]$)*
4. *$\langle a, \langle a, [\neg c] \rangle \rangle$ (resolution with the input clause $a \vee c$)*

5. $\langle a, \emptyset \rangle$ (merging of a, then truncation of $[\neg c]$)

Note that without the skipping rule, the derivation of a would not have been possible (with the model elimination strategy, the proof would stop at the second step). This is not really surprising since linear resolution strategies involving C-ordering are tailored for proof finding, and, as evoked previously, are not complete in consequence finding: the skipping rule is necessary to achieve this goal. In the case where Σ does not reduce to a clause but is any CNF formula, it is possible to take advantage of Proposition 3.24 so as to recover the simpler case where SOL-resolution can be used directly (i.e., Σ is a clause).

Proposition 3.25 *SOL-resolution is* complete in new characteristic consequence finding *in the sense that for every implicate γ of $\Phi \wedge \Sigma$ (where Φ is a CNF formula and Σ is a clause) that belongs to a stable production field $\mathcal{L}_T$ and is not an implicate of Φ, there exists an implicate of $\Phi \wedge \Sigma$ that belongs to $\mathcal{L}_T$, is not an implicate of Φ, is logically stronger than γ and has a $\mathcal{L}_T$-SOL-resolution proof from Φ and Σ.*

Example 3.45 *Let $\Phi = (a \vee b) \wedge (a \vee c)$ and $\Sigma = (\neg b \vee \neg c)$. Let $L = \{a, \neg c\}$. The $\langle L, \Phi \rangle$-implicate $(a \vee \neg c)$ of Σ w.r.t. Φ is a logical consequence of a clause (a) that has a $\langle L, \Phi \rangle$-SOL-resolution proof from Φ and Σ.*

Accordingly, $PI_L^{\Phi}(\Sigma)$ can be computed by taking the logically strongest clauses that have a $\langle L, \Phi \rangle$-SOL-resolution proof from Φ and Σ, while removing those that are logical consequences of Φ. It is worth noting that the set of logically strongest clauses that are derivable using SOL-resolution is not $PI_L(\Phi \wedge \Sigma)$ in the general case, but only a proper subset of it. This has two important consequences. On the one hand, using SOL-resolution for generating $\langle L, \Phi \rangle$-prime implicates does not come down to filter out the Φ-prime implicates of Σ among the L-prime implicates of $\Phi \wedge \Sigma$: SOL-resolution is actually oriented towards the production of $\langle L, \Phi \rangle$-implicates, hence many deduction checks w.r.t. Φ can be avoided, compared with the naive filtering approach. On the other hand, this shows that SOL-resolution is not complete in consequence finding for L-implicates: it is not the case that every L-implicate of $\Phi \wedge \Sigma$ is entailed by a clause that has a $\langle L, \Phi \rangle$-SOL-resolution proof from Φ and Σ (this is not ensured for the L-implicates of Φ):

Example 3.46 *Let $\Phi = (a \vee b) \wedge (a \vee \neg b)$ and $\Sigma = c$. Let $L = \{a\}$. There is no $\langle L, \Phi \rangle$-SOL-resolution proof from Φ with top clause Σ. Especially, the L-implicate a of $\Phi \wedge \Sigma$ cannot be derived using SOL-resolution from Φ and Σ.*

Accordingly, SOL-resolution cannot be used directly for computing L-prime implicates. Nevertheless, this difficulty can be easily circumvented by taking advantage of Proposition 3.17 and Proposition 3.24.

So as to determine whether or not a derived clause (that necessarily is a L-implicate of $\Phi \wedge \Sigma$) is a logical consequence of Φ, Inoue [Inoue, 1992] points out several strategies. Among them is the preliminary computation of $PI_L(\Phi)$ (which is used as a compilation). So as to derive $\langle L, \Phi \rangle$-implicates directly using SOL-resolution (avoiding an additional filtering step), it is possible to incorporate it in the notion of proof by requiring that no δ_i (the first projection of the i^{th} center clause of a SOL-resolution proof) is a logical consequence of Φ. Thus, undesirable clauses are removed as soon as they are generated (instead of being collected alltogether). Using this slight variant of SOL-resolution, Inoue shows that $PI_L^{\Phi}(\Sigma)$ can be computed directly as the set of the logically strongest clauses that are derivable from Φ and top clause Σ.

Other computational techniques can also be used so as to avoid generating many useless clauses and focus on the generation of the desired prime implicates. Thus, we have a distribution property for $\langle L, \Phi \rangle$–prime implicates.

Proposition 3.26 (distribution) *Let* $\Sigma_1, \ldots, \Sigma_k$ *be k formulas from* $PROP_{PS}$, Φ *be a formula from* $PROP_{PS}$ *and L be a finite subset of* L_{PS}. $PI_L^{\Phi}(\Sigma_1 \vee ... \vee \Sigma_k) = min(\{(\pi_{\Sigma_1} \vee ... \vee \pi_{\Sigma_k}); \pi_{\Sigma_1} \in PI_L^{\Phi}(\Sigma_1), ..., \pi_{\Sigma_k} \in PI_L^{\Phi}(\Sigma_k)$ *and* $\Phi \not\models (\pi_{\Sigma_1} \vee ... \vee \pi_{\Sigma_k})\}, \models)$.

Example 3.47 *Let* $\Phi = (a \Rightarrow b)$, $\Sigma_1 = a$, $\Sigma_2 = c$ *and* $L = \{a, c\}$. *We have* $PI_L^{\Phi}(\Sigma_1 \vee \Sigma_2) = \{(a \vee c)\}$, $PI_L^{\Phi}(\Sigma_1) = \{a\}$ *and* $PI_L^{\Phi}(\Sigma_2) = \{c\}$.

3.4 Complexity results

In this section, some complexity results about prime implicates, L–prime implicates, Φ–prime implicates and $\langle L, \Phi \rangle$–prime implicates are successively presented. As the metatheoretic properties given in Section 3.3 aim at explaining why some algorithms for computing prime implicates work (formally why they are correct), the following complexity results are expected to account for their performances *in the worst case*. More, they express some limitations to which any algorithm (including those that have not been designed yet) is submitted.

Starting with the basic notion of prime implicate, we successively consider L and Φ and both of them as additional parameters, and show whether or not they convey some additional source of complexity.

As we shall see in more details, computing prime implicates/implicants is hard in the general case, and this is also true for the more refined notions of L–prime implicate, Φ–prime implicate and the combination of them. Actually, two sources of intractability must be considered; on the one hand, the number of prime implicates of a formula is often very high; on the other hand, the basic problem of determining whether a given clause is a prime implicate of a formula is typically hard.

3.4.1 Prime implicates

The number of prime implicants/implicates of a propositional formula can easily be exponential in the size of the formula. Hence, every algorithm requires an exponential amount of space (hence, an exponential amount of time) to generate all the prime implicants/implicates of a formula in the worst case.

Example 3.48 *Let us consider a representation of the n-inputs parity function in so-called Reed-Muller normal form: $x_1 \oplus x_2 \oplus \ldots \oplus x_n$. This formula clearly admits 2^{n-1} prime implicates (and the same number of prime implicants).*

This is not the worst situation, however. Actually, the number of prime implicates/implicants of a propositional formula can be exponentially larger than the number of its models. This fact is well-known for a while [Fridshal, 1957] [Dunham and Fridshal, 1959]. Thus, Chandra and Markowsky [Chandra and Markowsky, 1978] exhibit families of n-variable boolean functions [23] having $\Omega(3^n/n)$ prime implicants/implicates. Igarashi [Igarashi, 1979] gives an improved lower bound. Beyond worst case situation, Provan [Provan, 1990] cites Kuznetsov [Kuznetsov, 1983], showing that the number of prime implicates of a n-variable boolean function is in $\Theta(2^n)$ for almost all boolean functions [24]. Chandra and Markowsky [Chandra and Markowsky, 1978] also show that there exist boolean functions represented by propositional formulas in CNF containing k clauses that admit more than $2^{\lfloor k/2 \rfloor}$ prime implicates.

Some upper bounds on the number of prime implicates/implicants of a propositional formula are also available. Especially, Chandra and Markowsky [Chandra and Markowsky, 1978] show that the number of prime implicates of a n–variable boolean function is in $\mathcal{O}(3^n/\sqrt{n})$ and bounded by 2^k if it is represented by the conjunction of k clauses (see also [McMullen and Shearer, 1986]).

Incremental techniques for computing prime implicates do not circumvent the problem in the worst case. Indeed, the number of prime implicates of $(\Sigma \wedge \pi)$ where π is a clause and Σ is given by its prime implicates can be exponential in the size of Σ [Kean and Tsiknis, 1990b].

[23] Let us recall that a boolean function can be considered as a propositional formula considered up to logical equivalence.

[24] This complexity result is not so dramatic because the boolean functions considered in practice are not good representatives of the whole family of all boolean functions. Especially, we know from Shannon's theorem [Shannon, 1949] that the great majority of n-variable boolean functions cannot be represented by a circuit containing a number of gates polynomial in n. As quoted by Papadimitriou [Papadimitriou, 1994], the fact that "*while we know that many exponentially difficult boolean functions exist, nobody has been able to come up with a natural family of boolean functions that requires more than a linear number of gates to compute*" is one of the major frustrations of complexity theory.

Example 3.49 *Let* $\Sigma_{k,m} = (p_1 \vee q_{11}) \wedge ... \wedge (p_1 \vee q_{1m}) \wedge (p_2 \vee q_{21}) \wedge ... \wedge (p_2 \vee q_{2m}) \wedge ... (p_k \vee q_{k1}) \wedge ... \wedge (p_k \vee q_{km})$. $\Sigma_{k,m}$ *has* $m * k$ *prime implicates (all the clauses belonging to the CNF of* $\Sigma_{k,m}$ *given above are such prime implicates). Let* π_k *be the clause* $\neg p_1 \vee ... \vee \neg p_k$. *Then* $\Sigma_{k,m} \wedge \pi_k$ *has* $(m+1)^k + m * k$ *prime implicates.*

However, it is worth noting that, under some very restricted conditions, an exponential blowup can be avoided. In particular, any Krom formula has a number of prime implicates bounded by a polynomial in its size. This can be viewed as a consequence of the correctness property of any resolution-based algorithm for computing prime implicates. Indeed, only a polynomial number of resolvents can be generated from a set of binary clauses [25]. In the same vein, monotone CNF formulas (i.e., CNF formulas in which every variable occurring positively does not occur negatively) are guaranteed to have only polynomially many prime implicates. Unfortunately, standard classes of formulas that are tractable for the satisfiability problem SAT (e.g., Horn, reverse Horn, renamable Horn classes) do not share the same property: a Horn (resp. reverse Horn, renamable Horn) formula can easily admit a number of prime implicates that is not polynomial in its size. It is also worth noting that DNF formulas, while tractable for SAT, may also possess exponentially many prime implicates. To be more precise, any DNF formula Σ containing k terms may have $\Omega(s^k)$ prime implicates where s is the minimum number of literals occurring in any term of Σ (this is an easy consequence of Proposition 3.9).

Beyond counting aspects, the problem of checking whether a clause is a prime implicate is computationally hard (but not "very hard" in the general case). Formally:

Definition 3.18 PRIME IMPLICATE *is the following decision problem:*

- *Given a formula* Σ *of* $PROP_{PS}$ *and a clause* π *of* $PROP_{PS}$*,*
- *Is* π *a prime implicate of* Σ*?*

This problem clearly is at least as hard as both satisfiability and deduction in propositional logic but remains feasible in polynomial time using two calls to an NP–oracle. To be more precise:

Proposition 3.27 PRIME IMPLICATE *is* BH_2*–complete.*

This complexity result shows in a precise way PRIME IMPLICATE as hard as satisfiability and deduction (i.e., both NP–hard and coNP–hard) but not much harder since in BH_2.

[25] Thus, resolution-based algorithms have a polynomial behaviour on such instances, while some other techniques, i.e., those based on the distribution property, may require exponential time.

Tractability can be achieved for several restrictions of PRIME IMPLICATE. Indeed, whenever Σ is tractable for clausal query answering (i.e., there exists an algorithm for checking whether or not $\Sigma \models \pi$ for every clause π in time polynomial in the input size), PRIME IMPLICATE belongs to P. This is a direct consequence of the way the algorithm with NP–oracle used to show PRIME IMPLICATE to belong to BH_2 works (see the proof in appendix). Thus, when Σ is Horn, reverse Horn, renamable Horn, PRIME IMPLICATE becomes tractable. This is also the case when Σ is given in DNF. In these cases, the function problem consisting in deriving, from Σ and a non-valid implicate π of it a prime implicate π_Σ of Σ s.t. $\pi_\Sigma \models \pi$ can be solved in polynomial time. In simple terms, extracting a prime implicate of Σ from Σ and an implicate of Σ is easy when Σ is tractable. The algorithm sketched in the proof of Proposition 3.27 can be adapted to this end in a straightforward way. This property is used in some algorithms for computing prime implicates, especially [Castell, 1996] [Castell, 1997]. It is also used in [Boros *et al.*, 1990] where it is shown that the problem of finding out all the prime implicates of a CNF formula Σ built up from a class of clauses tractable for SAT, stable for subsumption and for resolution is P–enumerable, i.e., there exists an algorithm for pointing out all prime implicates of Σ that is polynomial in the size of Σ plus the number of prime implicates of Σ.

It is easy to see that the complexity result given in Proposition 3.27 also applies to the following:

Definition 3.19 INCREMENTAL PRIME IMPLICATE *is the following decision problem:*

- *Given* Π *the set of prime implicates of a formula of* $PROP_{PS}$*, a formula* Σ *and a clause* π *of* $PROP_{PS}$*,*
- *Is* π *a prime implicate of* $\Pi \wedge \Sigma$*?*

Indeed, PRIME IMPLICATE is a restriction of INCREMENTAL PRIME IMPLICATE (obtained by assigning Π to $\{true\}$), so the BH_2–hardness follows. The membership to BH_2 is easy. Thus, the fact that Π is a set of prime implicates does not help in the general case. Nevertheless, it proves helpful in some restricted cases. Indeed, when Σ is given in DNF (especially when Σ is a clause or a term), the complexity of INCREMENTAL PRIME IMPLICATE falls down to P. Thus, provided that $\Sigma_1 \vee ... \vee \Sigma_k$ is a DNF of Σ, π is an implicate of $\Pi \wedge \Sigma$ iff $\Pi \models (\neg\Sigma_i \vee \pi)$ holds for every Σ_i ($i \in 1\ ..\ k$). Since $(\neg\Sigma_i \vee \pi)$ is a clause, the covering property shows how checking whether $\Pi \models (\neg\Sigma_i \vee \pi)$ holds or not can be achieved in time polynomial in the input size. The algorithm used to show PRIME IMPLICATE in BH_2 can then be used to check whether or not π is prime. Beyond this decision problem, the function problem consisting in computing the prime implicates of $\Pi \wedge \Sigma$ when Σ is a term (or more generally, a DNF formula consisting of k terms where k is a constant) can be solved in (deterministic) polynomial time (see Section 3.3.1).

3.4.2 L–prime implicates

Let us now investigate the complexity of L–prime implicates. Clearly enough, since $PI(\Sigma) = PI_{Lit(\Sigma)}(\Sigma)$, the complexity results related to the possibly exponential number of prime implicates of a formula still apply to L–prime implicates in the worst case. However, no formula can have more L-prime implicates than $\mathcal{O}(2^{card(L)})$ (simply because there are only $2^{card(L)}$ clauses generated from a set of literals L, up to logical equivalence).

Let us now consider the decision problems of checking whether or not a clause is a L–prime implicate:

Definition 3.20 L–PRIME IMPLICATE *is the following decision problem:*

- *Given a formula Σ of $PROP_{PS}$, a finite subset L of L_{PS} and a clause π of $PROP_{PS}$,*
- *Is π a L–prime implicate of Σ?*

It is not hard to show that:

Proposition 3.28 L–PRIME IMPLICATE *is* BH_2*–complete.*

Accordingly, checking whether a given clause is a L–prime implicate of a formula is hard in the general case. It is easy to see that the tractable restrictions pointed out in the previous section also apply to L–prime implicates. Thus, assuming that Σ belongs to a class of formulas that is tractable for clausal query answering (including e.g., Horn, reverse Horn, renamable Horn, Krom and DNF formulas), L–PRIME IMPLICATE belongs to P.

We have seen that, in contrast to the prime implicates situation, a formula Σ may have no L–prime implicates. We are now going to consider the following three questions from the point of view of computational complexity:

- Does a L–prime implicate of Σ exist?
- Does a L–prime implicate of Σ containing a given literal l exist?
- Is every L–prime implicate of Σ equivalent to a clause containing a given literal l?

All these three problems are hard in the general case:

Definition 3.21 L–PRIME IMPLICATE VACUITY *is the following decision problem:*

- *Given a formula* Σ *of* $PROP_{PS}$, *and a finite subset* L *of* L_{PS},
- *Is* $PI_L(\Sigma)$ *empty?*

Thanks to Proposition 3.11, it is easy to show that the set of L-prime implicates of Σ is empty iff every prime implicate of Σ contains a literal from $Lit(\Sigma) \setminus L$ [26]. As a consequence:

Proposition 3.29 L–PRIME IMPLICATE VACUITY *is* Π_2^p*–complete.*

Definition 3.22 L–PRIME IMPLICATE RELEVANCE *is the following decision problem:*

- *Given a formula* Σ *of* $PROP_{PS}$, *a finite subset* L *of* L_{PS}, *and a literal* l *from* L_{PS},
- *Does there exist a* L*–prime implicate of* Σ *that is equivalent to a clause containing* l*?*

Proposition 3.30 L–PRIME IMPLICATE RELEVANCE *is* Σ_2^p*–complete.*

We can note here that restricting the set of literals of the prime implicates under consideration is an additional source of complexity. Indeed, if this assumption were relaxed (i.e., L is not constrained), checking whether or not a prime implicate of a formula Σ containing a given literal l exists comes down to check whether or not Σ is dependent from l. It is quite easy to show that the latter problem is "only" NP–complete [Lang *et al.*, 1999], thus easier than the problem obtained when L is constrained since $\mathsf{NP} \subset \Sigma_2^p$ holds, unless the polynomial hierarchy collapses. Imposing some additional restrictions makes the relevance problem for L-prime implicates easier. Thus, if L is a satisfiable set of literals, the relevance problem becomes NP-complete; if Σ is a DNF formula, it becomes coNP-complete.

Definition 3.23 L–PRIME IMPLICATE NECESSITY *is the following decision problem:*

- *Given a formula* Σ *of* $PROP_{PS}$, *a finite subset* L *of* L_{PS}, *and a literal* l *from* L_{PS},
- *Is every* L*–prime implicate of* Σ *equivalent to a clause containing* l*?*

Proposition 3.31 L–PRIME IMPLICATE NECESSITY *is* Π_2^p*–complete.*

---

[26] We assume here without loss of generality that Σ is not valid and L is not empty. If Σ is valid, then *true* is its unique L–prime implicate (up to logical equivalence), hence $PI_L(\Sigma)$ is not empty. If $L = \emptyset$, then $PI_L(\Sigma)$ is empty iff Σ neither is valid, nor contradictory. Both cases can be handled in polynomial time using a NP–oracle.

3.4.3 Φ–prime implicates

Let us now turn to Φ–prime implicates. Once again, the complexity results related to the possibly exponential number of prime implicates in the worst case also apply to Φ–prime implicates since $PI(\Sigma) = PI^{true}(\Sigma)$ (when Σ is not valid).

We consider the following decision problem:

Definition 3.24 Φ–PRIME IMPLICATE *is the following decision problem:*

- *Given two formulas Φ and Σ of $PROP_{PS}$, and a clause π of $PROP_{PS}$,*
- *Is π a Φ–prime implicate of Σ?*

We can show that:

Proposition 3.32 Φ–PRIME IMPLICATE *is* BH_2*–complete.*

Actually, computing Φ–prime implicates for any formula Σ is just as difficult as computing Φ–prime implicates for literals Σ. Indeed, let *new* be a variable from $PS \setminus (Var(\Phi) \cup Var(\Sigma))$. Then it is easy to show that π is a Φ–prime implicate of a formula Σ iff π is a $\Phi \wedge (new \Rightarrow \Sigma)$–prime implicate of literal *new*, and π is not equivalent to *new*. Together with Proposition 3.21, this polynomial reduction from the general case to the specific case where Σ is a literal shows that computing the Φ–prime implicates of any Σ would be easy if computing the prime implicates of any formula Φ were easy (Proposition 3.18 shows it as well in a more direct way).

Clearly enough, tractable classes for PRIME IMPLICATE remain tractable classes for Φ–PRIME IMPLICATE; to be more precise, assuming that $\Phi \wedge \Sigma$ and Φ belong to the same class of formulas tractable for clausal deduction (including e.g., Horn, reverse Horn, renamable Horn, Krom and DNF formulas), Φ–PRIME IMPLICATE belongs to P. When Φ is given by its prime implicates, checking whether π is a prime implicate of Φ is easy. If it is not a prime implicate of Φ, π is a Φ–prime implicate of Σ iff it is a prime implicate of $\Phi \wedge \Sigma$. Since INCREMENTAL PRIME IMPLICATE is BH_2–complete, assuming that Φ is given by its prime implicates does not question the complexity of Φ–PRIME IMPLICATE in the general case. Nevertheless, it proves helpful in some restricted situations, especially when Σ is a DNF formula consisting of k terms (where k is a constant). Indeed, in this situation, the Φ–prime implicates of Σ can be computed in time polynomial in $|PI(\Phi)| + |\Sigma|$ (see Section 3.3.3). Subsequently, in this restricted case, Φ–PRIME IMPLICATE belongs to P.

Let us now turn to the problem of checking whether or not a Φ–prime implicate of Σ exists:

Definition 3.25 Φ–PRIME IMPLICATE VACUITY *is the following decision problem:*

- *Given two formulas Φ and Σ of $PROP_{PS}$,*
- *Is $PI^{\Phi}(\Sigma)$ empty?*

Clearly enough, this decision problem is equivalent to the problem of checking whether or not a Φ–implicate of Σ exists, i.e., we can find out a clause π s.t. $\Phi \wedge \Sigma \models \pi$ and $\Phi \not\models \pi$ hold. Every instance for which $\Phi \models \Sigma$ is a "no"-instance of the decision problem. Conversely, if there exists a model of Φ that is not a model of Σ, then there exists a model of $\neg\Sigma$ that is not a model of $\neg\Phi$. Then, the clause obtained by negating every literal of such a model of $\neg\Sigma$ is a logical consequence of Σ (hence a consequence of $\Phi \wedge \Sigma$) but not a logical consequence of Φ. Stated differently, this clause is a Φ–implicate of Σ. Thus, $PI^{\Phi}(\Sigma)$ is empty iff $\Phi \models \Sigma$ holds. Accordingly:

Proposition 3.33 Φ–PRIME IMPLICATE VACUITY *is* coNP*–complete.*

As an easy consequence of a result from [Selman and Levesque, 1990], a Φ–prime implicate of a literal Σ that is not equivalent to Σ can be exhibited in time polynomial in $|\Phi| + |\Sigma|$ when Φ is a Horn formula (provided that such a Φ–prime implicate exists, which can be also checked in polynomial time).

3.4.4 $\langle L, \Phi\rangle$–prime implicates

Let us finally consider the complexity of $\langle L, \Phi\rangle$–prime implicates. Taking L as an additional parameter neither changes the possible exponential number of prime implicates nor the complexity of recognizing them. Thus, in the worst case, a formula can have exponentially many $\langle L, \Phi\rangle$–prime implicates. Moreover, checking whether a given clause is a $\langle L, \Phi\rangle$–prime implicate of a formula Σ is hard in the general case.

Definition 3.26 $\langle L, \Phi\rangle$–PRIME IMPLICATE *is the following decision problem:*

- *Given two formulas Φ and Σ of $PROP_{PS}$, a finite subset L of L_{PS} and a clause π of $PROP_{PS}$,*
- *Is π a $\langle L, \Phi\rangle$–prime implicate of Σ?*

Proposition 3.34 $\langle L, \Phi\rangle$–PRIME IMPLICATE *is* BH_2*–complete.*

For the remaining decision problems we have considered, we have:

Definition 3.27 $\langle L, \Phi\rangle$–PRIME IMPLICATE VACUITY *is the following decision problem:*

- *Given two formulas Φ and Σ of $PROP_{PS}$, and a finite subset L of L_{PS},*
- *Is $PI_L^{\Phi}(\Sigma)$ empty?*

Proposition 3.35 $\langle L, \Phi\rangle$–PRIME IMPLICATE VACUITY *is Π_2^p–complete.*

Definition 3.28 $\langle L, \Phi\rangle$–PRIME IMPLICATE RELEVANCE *is the following decision problem:*

- *Given two formulas Φ and Σ of $PROP_{PS}$, a finite subset L of L_{PS}, and a literal l of L_{PS},*
- *Does there exist a $\langle L, \Phi\rangle$–prime implicate of Σ that is equivalent to a clause containing l?*

The relevance problem for $\langle L, \Phi\rangle$-prime implicates is just as hard as the relevance problem for L-prime implicates.

Proposition 3.36 $\langle L, \Phi\rangle$–PRIME IMPLICATE RELEVANCE *is Σ_2^p–complete.*

However, contrariwise to the L-prime implicates situation, imposing the constraint that L is a satisfiable set of literals does not lower the complexity of the relevance problem (it remains Σ_2^p-complete).

We have also identified the complexity of the necessity problem for $\langle L, \Phi\rangle$-prime implicates.

Definition 3.29 $\langle L, \Phi\rangle$–PRIME IMPLICATE NECESSITY *is the following decision problem:*

- *Given two formulas Φ and Σ of $PROP_{PS}$, a finite subset L of L_{PS}, and a literal l of L_{PS},*
- *Is every $\langle L, \Phi\rangle$–prime implicate of Σ equivalent to a clause containing l?*

Proposition 3.37 $\langle L, \Phi\rangle$–PRIME IMPLICATE NECESSITY *is Π_2^p–complete.*

Due to close connections with abduction (that will be clarified soon, see Section 4.2.2), these results, as well as many additional ones, can be recovered from the work by [Eiter and Gottlob, 1995]. In the case where Φ is Horn and Σ is restricted to a conjunction of positive literals, the complexity of these three decision problems are lowered by exactly one level in the polynomial hierarchy (see [Eiter and Gottlob, 1995] for details). Compared with the Φ–prime implicates situation, we can observe here that the restriction to a subset L of literals changes the complexity issues (especially, the complexity of the vacuity problem climbs up from the first to the second level of the polynomial hierarchy). Contrastingly, compared to the L–prime implicates situation, there is no change from the complexity point of view in the general case (loosely speaking, taking Φ into account is not a new source of complexity) for these decision problems.

4 Consequence Finding for Common Sense Reasoning

In this section, we show how consequence finding algorithms can prove helpful for various forms of common sense reasoning, especially abduction, simple default reasoning, circumscription and other forms of closed world reasoning. We also explain how clause management systems and related systems can be used to perform consequence finding for common sense reasoning.

4.1 Introduction

Many forms of logic-based common sense reasoning can be considered as deduction from a completed knowledge base. The additional knowledge to be taken into account typically depends of the prior knowledge, rendering inference from the completed knowledge base non-monotonic. Accordingly, such additional knowledge is hypothetical in essence (strengthening the original knowledge base may easily remove some of the assumptions made).

While different reasoning models can be characterized by different completion policies (roughly, those pieces of information that specify what are the theorems of the completed knowledge base), such policies can also be stated in many different ways. Very often, the additional knowledge to be taken into account is represented in an intensional way, i.e., in the metalanguage of the logic used to represent the prior knowledge, and the purpose is much more to characterize the formulas that must be considered true in the completed knowledge base than to precisely identify such additional knowledge; thus, some further inference rules can be used to characterize new theorems (as in default logic [Reiter, 1980]), or pre-orderings over the models of the knowledge base and a preference policy can be used to specify the worlds in which the new theorems holds (a preferred models semantics [Shoham, 1987] [Besnard and Siegel, 1988] can be achieved for various forms of non-monotonic inference).

An alternative way consists in deriving explicitly the additional knowledge to be considered. Such an approach to common sense reasoning is called *assumption-based reasoning*. Thus, assumption-based reasoning is centered around the various inferences that can be achieved whenever we are ready to assume some hypotheses being true. Within this approach, common sense reasoning from a knowledge base is achieved as classical reasoning (deduction) from a completed knowledge base. Consequence finding for common sense reasoning takes place in such a framework. As we shall see, consequence finding can be used so as to characterize the various assumptions that are conveyed through a non-standard interpretation of the knowledge base. Typically, such assumptions can also be characterized through a preferred models semantics.

Assumption-based reasoning has been pioneered by Poole [Poole, 1988] and further extended by [Brewka, 1989] [Cayrol, 1992] (mainly through the handling of additional preference criteria inside Poole's framework). In Poole's framework, the set of all possible individual hypotheses is given *a priori*, together with the knowledge base. Interestingly, assumption-based reasoning supports several forms of non-monotonic inference, including abduction, simple forms of default reasoning [Poole, 1988], and circumscription [Ginsberg, 1989] [Cayrol, 1992] without leaving the framework of classical (monotonic) logic. It is also closely related to *syntax-based belief base revision* [Nebel, 1992] (in this framework, the knowledge base is considered as an input, revision formula while the set of all individual hypotheses contains the (possibly contradictory) prior knowledge).

In the following, all these reasoning techniques are considered in a successive way. For each of them, after a brief presentation of some basic definitions, the focus is laid on their consequence finding characterization.

4.2 Abduction

4.2.1 Definitions

Abduction is inference to the best explanation. Several forms of abduction have been proposed so far, especially relational (or functional) abduction, logic-based abduction, probabilistic abduction. In the following, we focus only on logic-based abduction, which is the case if the available knowledge can be represented as a logical theory. This form of common sense inference proves helpful in many AI applications, including among others, diagnosis [Cox and Pietrzykowski, 1987] [Console *et al.*, 1989], natural language understanding [Hobbs *et al.*, 1993], planning, machine learning, etc.

Logic-based abductive inference is typically defined from an assumption-based theory:

Definition 4.1 (assumption-based theory) *Let Φ be a formula from $PROP_{PS}$ and H be a finite set of formulas from $PROP_{PS}$ (called* hypotheses *or, equivalently here,* assumptions*). The ordered pair $T = \langle \Phi, H \rangle$ is called an* assumption-based theory.

Σ represents the certain knowledge while H is the set of all individual hypotheses that are to be considered.

Given an assumption-based theory $T = \langle \Phi, H \rangle$, abductive inference aims at deriving an explanation for an observation report δ as a subset γ of H, that if it is added to Φ, enables δ to be derived as a logical consequence of the expanded theory. So as to avoid statements inconsistent with Φ to be viewed as explanations for every observation, only those γ which are consistent with Φ are authorized. Formally:

Definition 4.2 (abductive explanation) *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory and δ be a formula from $PROP_{PS}$. A (conjunctively-interpreted) subset γ of H is said an* abductive explanation *for δ w.r.t. T iff:*

- $\Phi \wedge \gamma \models \delta$, *and*
- $\Phi \wedge \gamma$ *is consistent.*

Example 4.1 *Let $T = \langle \Phi, H \rangle$ where $\Phi = (a \Rightarrow b) \wedge (c \Rightarrow d)$ and let $\delta = (b \wedge d)$. If $H = \{a, b, c, d, e\}$, then $(a \wedge c)$, $(a \wedge c \wedge e)$, $(b \wedge d)$ are abductive explanations for δ w.r.t. T. If $H = \{a, c\}$, then $(a \wedge c)$ is an abductive explanation for δ w.r.t. T.*

In the case where an abductive explanation for δ w.r.t. T exists, δ is said *explainable* within T; otherwise, it is said *unexplainable*. Clearly enough, every formula logically stronger than an unexplainable formula is unexplainable as well (w.r.t. the same assumption-based theory).

In the definition above, H is unrestricted. Especially, it is not required that H and the observation reports belong to disjoint languages. Indeed, it can be the case that a manifestation explains another manifestation. N-evertheless, in the general case, the notion of abductive explanation is too general for characterizing the best explanations for observation δ. Indeed, as a straightforward consequence of the monotonicity property of classical deduction, strengthening an abductive explanation for δ w.r.t. T results in an another, logically stronger, abductive explanation for δ w.r.t. T, as long as it remains consistent with Φ. Especially, the set of abductive explanations for δ w.r.t. T is huge (it would be infinite, would it be the case for H).

To deal with this difficulty, various preference criteria have been proposed so far in the literature, giving rise to various notions of *preferred abductive explanation*. Among these criteria are, e.g., *non triviality* (an abductive explanation γ for δ w.r.t. T is trivial if $\gamma \models \delta$ holds), *basicity* (an abductive explanation γ for δ w.r.t. T is basic if γ only has trivial explanation w.r.t. T) [Cox and Pietrzykowski, 1986]. *Minimality* (w.r.t. $\subseteq$) also is a commonly used preference criterion:

Definition 4.3 (minimal abductive explanation) *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory and δ be a formula from $PROP_{PS}$. A (conjunctively-interpreted) subset γ of H is said a* minimal abductive explanation *for δ w.r.t. T iff:*

- *γ is an abductive explanation for δ w.r.t. T, and*
- *for every abductive explanation γ' for δ w.r.t. T, if $\gamma' \subseteq \gamma$ holds, then $\gamma \subseteq \gamma'$ holds.*

Example 4.2 *Let $T = \langle \Phi, H \rangle$ where $\Phi = (a \Rightarrow b) \wedge (c \Rightarrow d)$ and let $\delta = (b \wedge d)$. If $H = \{a, b, c, d, e\}$, then $\{a, c\}$, $\{a, d\}$, $\{b, c\}$, $\{b, d\}$ are the minimal abductive explanations for δ w.r.t. T.*

Minimality enables removing every abductive explanation that is unnecessarily general. On the previous example, $\{a, c, e\}$ is considered an abductive explanation for δ w.r.t. T despite the fact that e is totally irrelevant to both δ and Φ (in the sense that it does not appear neither in δ nor in Φ). Contrastingly, $\{a, c, e\}$ is not a minimal abductive explanation for δ w.r.t. T since the abductive explanation $\{a, c\}$ for δ w.r.t. T is a proper subset of it. Thus, the set of minimal abductive explanations for δ w.r.t. T can be viewed as a compact characterization of the set of all abductive explanations for δ w.r.t. T in the sense that, for every explanation of the latter, there exists a minimal explanation of the former that is a superset of it (this is just a covering property for abductive explanations).

As we shall see in the following, logic-based abduction is closely related to several forms of non-monotonic inference, especially through the notion of consequence finding. It is also connected with deduction through predicate completion [Console *et al.*, 1991] and/or domain closure [Konolige, 1992].

From the point of view of computational complexity, propositional abduction is at least as difficult as propositional deduction in the general case (note that some tractable restrictions of logic-based abduction exist, see [Eshghi, 1993] [del Val, 1994] [Eiter and Gottlob, 1995] for details). Among the many decision problems that are related to abduction within assumption-based reasoning, Eiter and Gottlob [Eiter and Gottlob, 1995] address explainability (is δ explainable within T?), relevance of an individual hypothesis γ (is there exist a preferred abductive explanation for δ w.r.t. T that contains γ?), and necessity of an individual hypothesis γ (does every preferred abductive explanation for δ w.r.t. T contain γ?) for various preference criteria. They show the first two problems Σ_2^p–complete and the third one Π_2^p–complete in the general case when preferred explanations are the minimal ones. Eiter and Gottlob [Eiter and Gottlob, 1995] also investigate some other preference criteria and several restrictions of the three decision problems above from the point of view of computational complexity.

4.2.2 Consequence finding characterization

In order to characterize abduction as well as the other forms of assumption-based reasoning, it is convenient to assume that the assumption-based theory T under consideration is *normalized*, in the sense that the corresponding set H of hypotheses is composed only of variables of $Var(\Phi)$.

Interestingly enough, it is always possible to turn in polynomial time any assumption-based theory T into a normalized one, that is equivalent to T for the assumption-based reasoning tasks. Such a normalization process is called σ-*transformation* in [Kean and Tsiknis, 1990a] [Kean, 1992] [Kean and Tsiknis, 1993].

Definition 4.4 (σ-transformation) *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory, where $H = \{h_1, \ldots, h_n\}$. A σ-transformation of T is any assumption-based theory, noted $\sigma(T) = \langle \sigma(\Phi), \sigma(H) \rangle$ where:*

- $\sigma(\Phi) = \Sigma \wedge (h_1 \Leftrightarrow new_1) \wedge \ldots \wedge (h_n \Leftrightarrow new_n)$,
- $\sigma(H) = \{new_1, \ldots, new_n\}$,

and $\{new_1, \ldots, new_n\} \subseteq PS \setminus Var(\Phi)$.

Note that such a transformation mainly consists in giving names to the hypotheses of H under the form of new variables. Thus, new_i is the name given to the corresponding assumption h_i. Slightly abusing notations, we note $\sigma(h_i) = new_i$ for every i in $1 \,..\, n$, and for every subset γ of H, we note $\sigma(\gamma) = \{\sigma(h_i) | h_i \in \gamma\}$.

Clearly enough, $\sigma(T)$ is a normalized assumption-based theory. Moreover, it is equivalent to T for the determination of abductive explanations, in the following sense:

Proposition 4.1 *Let $T = \langle \Sigma, H \rangle$ be an assumption-based theory and δ be a formula from $PROP_{PS}$. γ is an abductive explanation for δ w.r.t. T iff $\sigma(\gamma)$ is an abductive explanation for δ w.r.t. $\sigma(T)$.*

Now, provided that $T = \langle \Sigma, H \rangle$ is a normalized assumption-based theory, the minimal explanations for δ w.r.t. Φ can be read off as the negations of the $\langle \neg H, \Phi \rangle$–prime implicates of $\neg\delta$:

Proposition 4.2 *Let $T = \langle \Phi, H \rangle$ be a normalized assumption-based theory and let δ be a formula from $PROP_{PS}$. γ is a minimal abductive explanation for δ w.r.t. Φ iff $\neg\gamma$ is a $\langle \neg H, \Phi \rangle$–prime implicate of $\neg\delta$.*

Example 4.3 *Let $T = \langle \Phi, H \rangle$ where $\Phi = (a \Rightarrow b) \wedge (c \Rightarrow d)$ and let $\delta = (b \wedge d)$. Let $H = \{a, b, c, d\}$. $PI^{\Phi}_{\neg L}(\neg\delta) = \{(\neg a \vee \neg c), (\neg a \vee \neg d), (\neg b \vee \neg c), (\neg b \vee \neg d)\}$. Accordingly, $\{a, c\}$, $\{a, d\}$, $\{b, c\}$, $\{b, d\}$ are the minimal abductive explanations for δ w.r.t. T.*

Clearly, all the computing techniques for deriving $\langle L, \Phi \rangle$–prime implicates that have been pointed out in Section 3.3.4 can be used to derive minimal abductive explanations. Moreover, we can observe that the complexity results that have been pointed out cohere with those derived by Eiter and Gottlob [Eiter and Gottlob, 1995].

4.3 Default reasoning

4.3.1 Definitions

Beyond abduction that is inference from observations to the hypotheses that explain them, one is typically interested in inference from hypotheses to their consequences. For such forms of inference within an assumption-based theory $\langle \Phi, H \rangle$, the main difficulty comes from the fact that hypotheses can be exclusive (i.e., all hypotheses cannot be true simultaneously in the general case) in the context of Φ, hence H itself cannot be considered as acceptable but several incompatible hypothetical belief sets must be considered.

Example 4.4 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory where $\Phi = (a \oplus b) \wedge (a \Rightarrow c) \wedge ((a \vee b) \Rightarrow d)$ and $H = \{a, \neg a, b, \neg b\}$. While it is easy to see that a and $\neg a$ cannot be simultaneously assumed true, this is also the case for a and b (and $\neg a$ and $\neg b$) within Φ. Accordingly, two different hypothetical belief sets can be considered here: the set of all the Φ–consequences of $\{a, \neg b\}$ and the set of all the Φ–consequences of $\{\neg a, b\}$.*

To state it in a more formal way, the notions of environment and interpretation are needed:

Definition 4.5 (environment, interpretation) *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory.*

- *An* environment *γ for T is a (conjunctively-interpreted) subset of H s.t. $\Phi \wedge \gamma$ is consistent,*
- *An* interpretation *γ of T is an environment γ for T s.t. every environment γ' for T that satisfies $\gamma \subseteq \gamma'$ also satisfies $\gamma' \subseteq \gamma$.*

We note $Interpretation(T)$ the set of all interpretations of T.

Example 4.5 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory where $\Phi = (a \oplus b) \wedge (a \Rightarrow c) \wedge ((a \vee b) \Rightarrow d)$ and $H = \{a, \neg a, b, \neg b\}$. T has two interpretations: $\{a, \neg b\}$ and $\{\neg a, b\}$.*

Note that there are no standard, commonly accepted, terms for denoting such notions. In the framework for default reasoning proposed in [Poole, 1988], our notion of environment is referred to as a *scenario* and an interpretation is a *maximal scenario*. In the ATMS framework [de Kleer, 1986], an environment is just a subset of H (i.e., an environment can be inconsistent with Φ, in which case it is sometimes referred to as a *conflict*). We are specifically interested in the minimal ones w.r.t. $\subseteq$:

Definition 4.6 (nogood) *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory. A* nogood *of T is a (conjunctively-interpreted) subset γ of H s.t. $\Phi \wedge \gamma$ is inconsistent and every subset γ' of H s.t. $\Phi \wedge \gamma'$ is inconsistent and $\gamma' \subseteq \gamma$ also satisfies $\gamma \subseteq \gamma'$. We note $Nogood(T)$ the set of all nogoods of T.*

Example 4.6 *Let* $T = \langle \Phi, H \rangle$ *be an assumption-based theory where* $\Phi = (a \oplus b) \wedge (a \Rightarrow c) \wedge ((a \vee b) \Rightarrow d)$ *and* $H = \{a, \neg a, b, \neg b\}$*.* T *has four nogoods:* $\{a, \neg a\}$*,* $\{b, \neg b\}$*,* $\{a, b\}$*, and* $\{\neg a, \neg b\}$*.*

Clearly enough, every assumption-based theory T has at least one interpretation whenever Φ is consistent, and more than one whenever it has a nogood containing at least two assumptions.

Nogoods and interpretations are related in the following way, using the notion of minimal hitting set (cf. [Reiter, 1987]):

Definition 4.7 (minimal hitting set) *Let* E *be a set and* C *a set of subsets of* E *(i.e.,* $C \subseteq 2^E$*). Let* I *be a subset of* E*.*

- I *is a* hitting set *of* C *iff* $\forall S \in C, I \cap S \neq \emptyset$*.*
- I *is a* minimal hitting set *of* C *iff* I *is a hitting set of* C *and no proper subset of* I *is a hitting set of* C*.*

We note $MHS(C)$ *the set of all the minimal hitting sets of* C*.*

For any subset γ of H, we note $\bar{\gamma} = \{h \in H | h \notin \gamma\}$ the complement of γ into H.

Proposition 4.3

- *The complements into* H *of the minimal hitting sets of the nogoods of* T *are exactly its interpretations:* $Interpretation(T) = \{\bar{\gamma} \subseteq H | \gamma \in MHS(Nogood(T))\}$*.*
- *The minimal hitting sets of the complements into* H *of the interpretations of* T *are exactly its nogoods:* $Nogood(T) = MHS(\{\bar{\gamma} \subseteq H | \gamma \in Interpretation(T)\}$*.*

Interestingly enough, assumption-based theories can direcly serve as a formal framework for syntax-based belief revision. Indeed, given a set H of pieces of prior knowledge and an input, revision formula Φ, the set of all so-called *maximal consistent subbases* of H w.r.t. Φ, i.e., the subsets of H that are consistent with Φ and maximal w.r.t. $\subseteq$, is exactly $Interpretation(T)$. Furthermore, the set of all so-called *minimal inconsistent subbases* of H w.r.t. Φ, i.e., the subsets of H that are inconsistent with Φ and are minimal w.r.t. $\subseteq$, is exactly $Nogood(T)$.

The belief set that is obtained by assuming that an interpretation holds within Φ is referred to as an extension of T. Formally:

Definition 4.8 (extension (of an assumption-based theory)) *Let* $T = \langle \Phi, H \rangle$ *be an assumption-based theory. An* extension E_γ *of* T *is the logical closure of the formula* $\Phi \wedge \gamma$*, where* γ *is an interpretation of* T*.*

When E_γ is the deductive closure of $(\Phi \wedge \gamma)$ for an interpretation γ of T, E_γ is said the extension of T generated by γ and γ is referred to as the *generating environment* for extension E_γ.

Example 4.7 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory where $\Phi = (a \oplus b) \wedge (a \Rightarrow c) \wedge ((a \vee b) \Rightarrow d)$ and $H = \{a, \neg a, b, \neg b\}$. c and d belong to the extension of T generated by $\{a, \neg b\}$, d belongs to the extension of T generated by $\{\neg a, b\}$ but this is not the case for c.*

In the ATMS terminology, the extension generated by γ is called the *context* of γ.

Interestingly, the notion of extension within assumption-based reasoning can be considered as a specific case of extension in the sense of Reiter's default reasoning [Reiter, 1980]. In order to present it in more formal terms, we first need some materials from default reasoning[27] (restricted to the propositional case which is sufficient for our purpose):

Definition 4.9 (default theory) *A* default theory *T is an ordered pair $\langle \Phi, D \rangle$, where Φ is a formula from $PROP_{PS}$ and D is a finite set of inference rules, called* defaults. *Each default d of D is a triple $\langle \alpha_d, \beta_{d_1} \ldots \beta_{d_n}, \gamma_d \rangle$, where α_d is a formula from $PROP_{PS}$ called the* prerequisite *of d, β_{d_1}, ..., β_{d_n} are formulas from $PROP_{PS}$ that form the* justification *of d and γ_d is a formula from $PROP_{PS}$ called the* consequence *of d.*

Default rules enable non-monotonic inference to be achieved. Intuitively, in any state of knowledge where the prerequisite of a default rule is satisfied and the associated justification is not contradicted, the consequence can be derived. Since defaults may easily have conflicting consequences, several belief sets corresponding to the consequences of a default theory may exist. They are also called extensions. Since care must be taken that none of the justifications of the defaults used in the construction of an extension conflicts with its final contents, the formal definition of an extension of a default theory is rather complex [Reiter, 1980]:

Definition 4.10 (extension (of a default theory)) *Let $T = \langle \Phi, D \rangle$ be a default theory and let Σ be any formula from $PROP_{PS}$. Let $\Gamma(\Sigma)$ be the smallest (w.r.t. set-inclusion) subset of $PROP_{PS}$ s.t.:*

- *$\Psi \in \Gamma(\Sigma)$, and*
- *$\Gamma(\Sigma)$ is deductively closed, and*
- *for every $d = \langle \alpha_d, \beta_{d_1}, \ldots, \beta_{d_n}, \gamma_d \rangle \in D$, if $\alpha_d \in \Gamma(\Sigma)$ and $\Sigma \wedge \beta_{d_i}$ is consistent for every $i \in 1 \,..\, n$, then $\gamma_d \in \Gamma(\Sigma)$.*

[27] More details about default reasoning can be found in [Reiter, 1980] [Besnard, 1989].

Then a subset E of $PROP_{PS}$ is an extension of T iff it is a fixed point of Γ, i.e., $\Gamma(E) = E$ holds.

Thus, within default reasoning, a formula can be considered true when it is the consequence of a default, as long as the corresponding prerequisite is satisfied and the corresponding justification does not contradict the current state of beliefs. This contrasts with what happens within assumption-based reasoning where an assumption can be considered true as soon as it does not contradict the knowledge base. There is nevertheless a close connection between both cases: the latter actually is a restriction of the former.

Proposition 4.4 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory, where $H = \{\gamma_1, ..., \gamma_n\}$. E is an extension of T iff E is an extension of the default theory $\langle \Phi, D_H \rangle$, where $D_H = \{\langle true, \gamma_1, \gamma_1 \rangle, ..., \langle true, \gamma_n, \gamma_n \rangle\}$.*

The default rule $\langle true, \gamma_i, \gamma_i \rangle$ corresponding to assumption γ_i is called a *supernormal default*, i.e., a *unary normal default* (the justification coincides with the consequence) *without prerequisite* (the prerequisite is valid). Accordingly, any assumption-based theory can be considered as a prerequisite-free normal default theory and the converse also holds.

Poole [Poole, 1988] argues that restricting default logic to prerequisite-free normal default logic is sufficient for many purposes. In particular, while a (general) default theory may admit zero, one or more extensions in the general case, normal default theories always have an extension [Reiter, 1980].

Within assumption-based reasoning (and more generally within default reasoning), we are typically interested in knowing whether or not a given formula Ψ belongs to an extension of T or to all the extensions of T. Considering that Ψ holds when it belongs to one extension of T is usually called *credulous or brave reasoning.* Conversely, considering that Ψ holds when it belongs to every extension of T is usually called *skeptical or safe reasoning.* Formally:

Definition 4.11 (credulous reasoning, skeptical reasoning) *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory and Ψ be a formula from $PROP_{PS}$.*

- *Ψ is a* credulous consequence *of T, noted $T \vdash_{credulous} \Psi$ iff Ψ belongs to at least one extension of T.*
- *Ψ is a* skeptical consequence *of T, noted $T \vdash_{skeptical} \Psi$ iff Ψ belongs to every extension of T.*

Example 4.8 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory where $\Phi = (a \oplus b) \wedge (a \Rightarrow c) \wedge ((a \vee b) \Rightarrow d)$ and $H = \{a, \neg a, b, \neg b\}$. c is a credulous consequence of T and d is a skeptical consequence of T.*

Obviously, every skeptical consequence of T also is a credulous one (assuming that Φ is satisfiable), but is not necessarily a logical consequence of Φ. Indeed, skeptical reasoning does not reduce to deduction in the general case (on the example above, while d belongs to every extension of T, it is not entailed by Φ).

Interestingly, both credulous and skeptical reasoning can be given a preferred model semantics, based on the notion of H-maximal model:

Definition 4.12 (H-maximal model) *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory. Let I and J be two models of Φ. We write $I \leq_H J$ iff for every formula γ from H, if $I \models \gamma$, then $J \models \gamma$. $\leq_H$ is a partial pre-ordering over the models of Φ and every model of Φ that is maximal w.r.t. $\leq_H$ is called a H-*maximal model *of Φ.*

Clearly enough, if I is a H-maximal model of Φ, then the set of formulas γ from H that are satisfied by I is an interpretation of T. Conversely, for every interpretation γ of T, there exists a H-maximal model I of Φ s.t. $\gamma = \{h \in H | I \models h\}$. As a consequence:

Proposition 4.5 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory and Ψ be a formula from $PROP_{PS}$.*

- *$T \vdash_{credulous} \Psi$ iff Ψ is true in at least one H-maximal model of Φ.*
- *$T \vdash_{skeptical} \Psi$ iff Ψ is true in every H-maximal model of Φ.*

As we shall see in the following, there are close connections between both credulous and skeptical reasoning from an assumption-based theory and abduction and some forms of circumscription. Especially, through their abductive characterizations, we shall show in the next section how credulous and skeptical reasoning can be implemented using consequence finding techniques. The complexity of prerequisite-free normal default logic has been investigated in [Selman and Levesque, 1990] [Gottlob, 1992] [Stillman, 1992]. In the general case, checking whether a formula is a credulous consequence of an assumption-based theory is Σ_2^p-complete, while checking whether a formula is a skeptical consequence of an assumption-based theory is Π_2^p-complete. Thus, the loss of expressivity that results from considering prerequisite-free normal default logic instead of full default logic is not balanced by a gain in complexity. To be more precise, even if some tractable restrictions exist (see [Kautz and Selman, 1991] [Stillman, 1992]), credulous (resp. skeptical) reasoning in full (propositional) default logic is just as hard as credulous (resp. skeptical) reasoning in prerequisite-free normal default logic [Gottlob, 1992] [Stillman, 1992]. Contrastingly, the existence of an extension is always ensured in prerequisite-free normal default logic, while deciding it in a general default theory is computationally hard (it is Σ_2^p–complete for semi-normal default theories [Gottlob, 1992] [Stillman, 1992]).

4.3.2 Consequence finding characterization

Let us first explain how interpretations and nogoods can be computed using consequence finding algorithms. Here again it is convenient to assume that the assumption-based theory T under consideration is normalized (cf. Section 4.2.2). This can be done without loss of generality:

Proposition 4.6 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory, and let γ be a subset of H. Let $\sigma(T)$ be any σ-transformation of T:*

- *γ is a nogood of T iff $\sigma(\gamma)$ is a nogood of $\sigma(T)$.*
- *γ is an interpretation of T iff $\sigma(\gamma)$ is an interpretation of $\sigma(T)$.*

As a direct corollary to Proposition 3.12, it is easy to show that, in any normalized assumption-based theory $T = \langle \Phi, H \rangle$, it is always possible to forget every literal except those of H, without modifying the nogoods and the interpretations. Formally, we have $Nogood(T) = Nogood(\langle forget(\Phi, Lit(\Phi) \setminus H), H \rangle)$ and $Interpretation(T) = Interpretation(\langle forget(\Phi, Lit(\Phi) \setminus H), H \rangle)$.

We have the following characterization of nogoods in terms of L-prime implicates:

Proposition 4.7 *Let $T = \langle \Phi, H \rangle$ be a normalized assumption-based theory. $Nogood(T) = \{Lit(\gamma) | \neg\gamma \in PI_{\neg H}(\Phi)\}$.*

Example 4.9 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory where $\Phi = (a \oplus b) \wedge (a \Rightarrow c) \wedge ((a \vee b) \Rightarrow d)$ and $H = \{a, b\}$. $\{a, b\}$ is the unique nogood of T since $PI_{\neg H}(\Phi) = \{(\neg a \vee \neg b)\}$ (up to logical equivalence).*

Note that the fact that H is a consistent set of literals (which is ensured by the normalization condition) is important. The proposition would not hold otherwise:

Example 4.10 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory where $\Phi = (a \oplus b) \wedge (a \Rightarrow c) \wedge ((a \vee b) \Rightarrow d)$ and $H = \{a, \neg a, b, \neg b\}$. $PI_{\neg H}(\Phi) = \{(a \vee b), (\neg a \vee \neg b)\}$. Accordingly, the nogoods $\{a, \neg a\}$ and $\{b, \neg b\}$ have been lost.*

The problem lies in the fact that $\models$ and $\supseteq$ coincide on (conjunctively-interpreted) sets of literals, only when they are consistent (hence, this problem does not occur when sets of literals representing abductive explanations or interpretations are considered, because they are -by definition- consistent).

Thanks to Proposition 4.7, it is possible to take advantage of Proposition 4.3 to derive $Interpretation(T)$ from $Nogood(T)$. Let us now show how credulous and skeptical reasoning from an assumption-based theory can be implemented using consequence finding, through their abductive characterizations:

Proposition 4.8 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory. Let E_γ be the extension of T generated by interpretation γ. Let Ψ be a formula from $PROP_{PS}$. $\Psi \in E_\gamma$ holds iff there exists a (minimal) abductive explanation γ' for Ψ w.r.t. T s.t. $\gamma \models \gamma'$ holds.*

Example 4.11 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory where $\Phi = (a \oplus b) \wedge (a \Rightarrow c) \wedge ((a \vee b) \Rightarrow d)$ and $H = \{a, \neg a, b, \neg b\}$. c belongs to the extension of T generated by $\{a, \neg b\}$ since a is a minimal abductive explanation for c w.r.t. T.*

We obtain immediately the following corollary:

Corollary 4.1 (credulous reasoning) *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory. Let Ψ be a formula from $PROP_{PS}$. $T \vdash_{credulous} \Psi$ iff Ψ is explainable within T.*

Provided that generating environments γ for all the extensions of T have been computed, Proposition 4.8 also shows how the membership of a given formula Ψ to all these extensions can be performed: it is sufficient to check that for every interpretation γ, there exists a minimal abductive explanation γ' for Ψ w.r.t. T s.t. $\gamma \models \gamma'$ holds.

Now, the next proposition shows how the membership of Ψ to every extension of T can be checked through abductive explanations, without making any explicit reference to the extensions of T themselves:

Proposition 4.9 (skeptical reasoning) *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory. Let Ψ be a formula from $PROP_{PS}$. Let $\{\gamma_1, ..., \gamma_k\}$ be the set of all minimal abductive explanations for Ψ w.r.t. T. $T \vdash_{skeptical} \Psi$ iff $(\neg\gamma_1 \wedge ... \wedge \neg\gamma_k)$ is unexplainable within T.*

$(\neg\gamma_1 \wedge ... \wedge \neg\gamma_k)$ is equivalent to the negation of the logically weakest formula Σ s.t. $Lit(\Sigma) \subseteq H$, $\Phi \wedge \Sigma$ is consistent and $\Phi \wedge \Sigma \models \Psi$ holds. In the case where H is closed under negation, the proposition above can be simplified to Ψ belongs to every extension of T iff $\Phi \models (\gamma_1 \vee ... \vee \gamma_k)$ holds. Indeed, in this restricted situation, every term built up from H that is consistent with Φ is a trivial explanation for itself w.r.t. T.

Example 4.12 *Let $T = \langle \Phi, H \rangle$ be an assumption-based theory where $\Phi = (a \oplus b) \wedge (a \Rightarrow c) \wedge ((a \vee b) \Rightarrow d)$ and $H = \{a, \neg a, b, \neg b\}$. d belongs to every extension of T since the set of all minimal abductive explanations for d w.r.t. T is $\{a, b\}$ and $(a \vee b)$ is a logical consequence of Φ. Contrastingly, c does not belong to every extension of T since the set of all minimal abductive explanations for c w.r.t. T is $\{a\}$ and a is not a logical consequence of Φ.*

Such translations of credulous and skeptical reasoning into an abductive framework prove helpful from a computational point of view (every algorithm used for generating minimal abductive explanations can also be used to solve both problems), as well as from the point of view of computational complexity (Corollary 4.1 gives a polynomial reduction from the problem of checking whether a given formula holds in some extension of T to the problem of checking whether it has an abductive explanation w.r.t. T). Especially, the consequence finding techniques presented in Section 3.3.4 can be directly used to achieve the reasoning tasks described above through Proposition 4.2.

4.4 Circumscription and closed world reasoning

4.4.1 Definitions

In many knowledge bases (especially, in simple relational databases), negative information is typically not represented in an explicit way. Indeed, considering that every relevant piece of positive information can be found in the knowledge base, every positive fact that cannot be deduced from it is assumed false. This is the basic idea of closed world reasoning. From a logical point of view, such kind of inference is non-monotonic since expanding the knowledge base with new information does not necessarily lead to an increase of its deductive closure (some previously derived theorems may easily be questioned).

There are several ways to characterize closed world reasoning. In particular, closed world reasoning can be characterized by eliciting the pieces of knowledge that must added to the original knowledge base to obtain the corresponding completed knowledge base; in simpler words, closed world reasoning can be achieved through assumption-based reasoning. The negations of the formulas that must be added to the original knowledge base to complete it are called *free for negation formulas* [Gelfond *et al.*, 1986]. Accordingly, closed world reasoning can be considered as deductive inference from the original knowledge base completed by the negations of its free for negation formulas (which vary according to the completion policy under consideration).

Closed world reasoning gathers many non-monotonic techniques for inference based on the lack of some information in the knowledge base. The simplest form of closed world reasoning that has been pointed out is Reiter's *closed world assumption* [Reiter, 1978]. Under this assumption (that is represented in the metalanguage of the logic), every positive (ground) fact that is not a logical consequence of the knowledge base is assumed false in it.

Definition 4.13 (closed world assumption) *Let Φ, Ψ be formulas from*

$PROP_{PS}$. Ψ *is* free for negation w.r.t. the closed world assumption *(CWA-ffn for short) given* Φ *iff* Ψ *is a positive literal s.t.* $\Phi \not\models \Psi$ *holds.*

Example 4.13 *Let* Φ *be the formula* $((\neg a \wedge b) \Rightarrow c) \wedge b$. *Both* a *and* c *are CWA-ffn given* Φ.

Definition 4.14 (closed world reasoning) *Let* Φ, Σ *be formulas from* $PROP_{PS}$. Σ *is a* consequence of Φ interpreted under the closed world assumption, *noted* $\Phi \vdash_{CWA} \Sigma$ *iff* $\Phi \wedge \{\neg\Psi; \Psi$ *is CWA-ffn given* $\Phi\} \models \Sigma$ *holds.*

Clearly enough, this form of closed world reasoning is very basic. In particular, closed world assumption may generate inconsistency as in the example above (this is the case whenever the knowledge base contains some disjunctive positive information, e.g., $a \vee c$ is a logical consequence of Φ).

Based on this consideration, Minker [Minker, 1982] proposes a more sophisticated form of closed world reasoning, called *generalized closed world assumption*:

Definition 4.15 (generalized closed world assumption) *Let* Φ, Ψ *be formulas from* $PROP_{PS}$. Ψ *is* free for negation w.r.t. the generalized closed world assumption *(GCWA-ffn for short) given* Φ *iff* Ψ *is a positive literal, and for every positive clause* γ, *if* $\Phi \not\models \gamma$ *holds, then* $\Phi \not\models \gamma \vee \Psi$ *holds as well.*

Example 4.14 *Let* Φ *be the formula* $((\neg a \wedge b) \Rightarrow c) \wedge b$. *There is no GCWA-ffn formula given* Φ. *Hence, no assumption is to be considered when* Φ *is interpreted under the generalized closed world assumption.*

Definition 4.16 (generalized closed world reasoning) *Let* Φ, Σ *be formulas from* $PROP_{PS}$. Σ *is a* consequence of Φ interpreted under the generalized closed world assumption, *noted* $\Phi \vdash_{GCWA} \Sigma$ *iff* $\Phi \wedge \{\neg\Psi; \Psi$ *is GCWA-ffn given* $\Phi\} \models \Sigma$ *holds.*

Several refinements of this approach have then been defined. Thus, an extension of the generalized closed world assumption is the *extended generalized closed world assumption* by Yahya and Henschen [Yahya and Henschen, 1985]. In contrast to the generalized closed world assumption, conjunction of positive literals are allowed as free for negation formulas:

Definition 4.17 (extended generalized closed world assumption) *Let* Φ, Ψ *be formulas from* $PROP_{PS}$. Ψ *is* free for negation w.r.t. the extended generalized closed world assumption *(EGCWA-ffn for short) given* Φ *iff* Ψ *is a conjunction of positive literals, and for every positive clause* γ, *if* $\Phi \not\models \gamma$ *holds, then* $\Phi \not\models \gamma \vee \Psi$ *holds as well.*

Example 4.15 *Let Φ be the formula $((\neg a \wedge b) \Rightarrow c) \wedge b$. $(a \wedge c)$ is EGCWA-ffn given Φ. No assumption different from $\neg(a \wedge c)$ is to be considered when Φ is interpreted under the extended generalized closed world assumption.*

Definition 4.18 (extended generalized closed world reasoning) *Let Φ, Σ be formulas from $PROP_{PS}$. Σ is a* consequence of Φ interpreted under the extended generalized closed world assumption, *noted $\Phi \vdash_{EGCWA} \Sigma$ iff $\Phi \wedge \{\neg\Psi; \Psi$ is EGCWA-ffn given $\Phi\} \models \Sigma$ holds.*

Gelfond and Przymusinska [Gelfond and Przymusinska, 1986] introduced another refinement of the generalized closed world assumption as the *careful closed world assumption*, obtained by applying the generalized closed world assumption to a specified set of atoms (instead of every atom). Thus, the propositional symbols occurring in Φ are partitioned into three sets: P, Q, and Z. P is the set of atoms that we prefer being false, such atoms are called *minimized*. Q is the set of *fixed* atoms, those that are not allowed to vary when trying to falsify the atoms of P. The remaining atoms (those of Z) are called *varying* to state that they are allowed to vary when trying to falsify the atoms of P.

Definition 4.19 (careful closed world assumption) *Let Φ be a formula from $PROP_{PS}$ and $\langle P, Q, Z\rangle$ a partition of $Var(\Phi)$. Let Ψ be a formula from $PROP_{PS}$. Ψ is* free for negation w.r.t. the careful closed world assumption *(CCWA-ffn for short) given Φ and $\langle P, Q, Z\rangle$ iff Ψ is a literal from L_P^+, and for every clause γ s.t. $Lit(\gamma) \subseteq L_P^+ \cup L_Q$, if $\Phi \not\models \gamma$ holds, then $\Phi \not\models \gamma \vee \Psi$ as well.*

Example 4.16 *Let Φ be the formula $((\neg a \wedge b) \Rightarrow c) \wedge b$. Let $P = \{a\}$, $Q = \{b\}$ and $Z = \{c\}$. a is CCWA-ffn given Φ and $\langle P, Q, Z\rangle$. No assumption different from $\neg a$ is to be considered when Φ is interpreted under the careful closed world assumption.*

Definition 4.20 (careful closed world reasoning) *Let Φ, Σ be formulas from $PROP_{PS}$. Let $\langle P, Q, Z\rangle$ be a partition of $Var(\Phi)$. Σ is a* consequence of Φ interpreted under the careful closed world assumption *given $\langle P, Q, Z\rangle$, noted $\Phi \vdash_{CCWA}^{\langle P,Q,Z\rangle} \Sigma$ iff $\Phi \wedge \{\neg\Psi; \Psi$ is CCWA-ffn given Φ and $\langle P, Q, Z\rangle\} \models \Sigma$ holds.*

In contrast to both the closed world assumption, the generalized closed world assumption, and the extended generalized closed world assumption, the careful closed world assumption enables new positive facts (i.e., positive facts that are not consequence of Φ) to be derived. Thus, in the example above, we have $\Phi \vdash_{CCWA}^{\langle P,Q,Z\rangle} c$, while $\Phi \not\models c$ holds.

Finally, the most sophisticated form of closed world reasoning that has been proposed so far is the *extended closed world assumption* by Gelfond, Przymusinska and Przymusinski [Gelfond *et al.*, 1989]. It includes both features of the careful closed world assumption and of the extended generalized closed world assumption. Indeed:

Definition 4.21 (extended closed world assumption) *Let Φ be a formula from $PROP_{PS}$ and $\langle P, Q, Z \rangle$ a partition of $Var(\Phi)$. Let Ψ be a formula from $PROP_{PS}$. Ψ is* free for negation w.r.t. the extended closed world assumption *(ECWA-ffn for short) given Φ and $\langle P, Q, Z \rangle$ iff $Var(\Psi) \cap Z = \emptyset$, and for every clause γ s.t. $Lit(\gamma) \subseteq L_P^+ \cup L_Q$, if $\Phi \not\models \gamma$ holds, then $\Phi \not\models \gamma \vee \Psi$ as well.*

Example 4.17 *Let Φ be the formula $((\neg a \wedge b) \Rightarrow c) \wedge b$. Let $P = \{a\}$, $Q = \{b\}$ and $Z = \{c\}$. a is ECWA-ffn given Φ and $\langle P, Q, Z \rangle$. No assumption different from $\neg a$ is to be considered when Φ is interpreted under the extended closed world assumption.*

Definition 4.22 (extended closed world reasoning) *Let Φ, Σ be formulas from $PROP_{PS}$. Let $\langle P, Q, Z \rangle$ be a partition of $Var(\Phi)$. Σ is a* consequence of Φ interpreted under the extended closed world assumption *given $\langle P, Q, Z \rangle$, noted $\Phi \vdash_{ECWA}^{\langle P,Q,Z \rangle} \Sigma$ iff $\Phi \wedge \{\neg \Psi; \Psi$ is ECWA-ffn given Φ and $\langle P, Q, Z \rangle\} \models \Sigma$ holds.*

New positive facts are also derivable from a knowledge base interpreted under the extended closed world assumption. Interestingly, in the propositional case, extended closed world assumption is equivalent to (parallel predicate) *circumscription*, the well-known general form of common sense reasoning proposed by McCarthy [McCarthy, 1986]. In contrast to the previous definition of circumscription given in [McCarthy, 1980], varying atoms are allowed; this is important since the capacity of taking into account such atoms is mandatory for deriving new positive facts.

Sophisticated approaches to closed world reasoning have preferred models characterizations, based on the notion of *minimal model* [Lifschitz, 1985]:

Definition 4.23 (minimal model) *Let Φ be a formula from $PROP_{PS}$ and $\langle P, Q, Z \rangle$ a partition of $Var(\Phi)$. Let I and J be two models of Φ. We write $I \leq_{\langle P,Q,Z \rangle} J$ iff I and J assign the same truth value to the atoms of Q, and the set of atoms of P interpreted as 1 in I is a subset of the set of atoms of P interpreted as 1 in J. $\leq_{\langle P,Q,Z \rangle}$ is a partial pre-ordering over the models of Φ, and every model of Φ that is minimal w.r.t. $\leq_{\langle P,Q,Z \rangle}$ is called a $\langle P, Q, Z \rangle$-*minimal model *of Φ.*

For simplicity, $\langle Var(\Phi), \emptyset, \emptyset \rangle$-minimal models of Φ are referred to as its minimal models. They correspond to the case where every atom is preferred false.

Proposition 4.10 *Let Φ be a formula from $PROP_{PS}$ and $\langle P, Q, Z\rangle$ a partition of $Var(\Phi)$. Let Ψ be a formula from $PROP_{PS}$.*

- *Ψ is GCWA-ffn given Φ iff Ψ is a positive literal and for every minimal model M of Φ, we have $M \not\models \Psi$.*
- *Ψ is EGCWA-ffn given Φ iff Ψ is a conjunction of positive literals and for every minimal model M of Φ, we have $M \not\models \Psi$.*
- *Ψ is CCWA-ffn given Φ and $\langle P, Q, Z\rangle$ iff Ψ is a literal of L_P^+ and for every $\langle P, Q, Z\rangle$-minimal model M of Φ, we have $M \not\models \Psi$.*
- *Ψ is ECWA-ffn given Φ and $\langle P, Q, Z\rangle$ iff $Var(\Psi) \cap Z = \emptyset$ and for every $\langle P, Q, Z\rangle$-minimal model M of Φ, we have $M \not\models \Psi$.*

This proposition clearly shows that the generalized closed world assumption can be considered as a specific case of the careful closed world assumption, achieved by minimizing every atom of Φ (i.e., $P = Var(\Phi)$). In the same vein, the extended generalized closed world assumption is the restriction of extended closed world assumption obtained by minimizing every atom of Φ. Finally, it is worth noting that it is possible to remove the fixed atoms [de Kleer and Konolige, 1989] as well as the varying ones [Cadoli *et al.*, 1992] from a circumscription policy, and this can prove computationally helpful in some situations.

There are several formal connections between closed world reasoning and other forms of non-monotonic inference. Especially, the following proposition shows how inference from a knowledge base interpreted under the extended closed world assumption amounts to skeptical reasoning from an assumption-based theory [Etherington, 1987] [Inoue and Helft, 1990].

Proposition 4.11 *Let Φ be a formula from $PROP_{PS}$ and $\langle P, Q, Z\rangle$ a partition of $Var(\Phi)$. Let Ψ be a formula from $PROP_{PS}$. $\Phi \vdash_{ECWA}^{\langle P,Q,Z\rangle} \Psi$ iff Ψ is a skeptical consequence of the assumption-based theory $\langle \Phi, L_P^- \cup L_Q\rangle$.*

Accordingly, assumption-based reasoning techniques can be used to implement circumscription. We shall return to it in the next section.

The complexity of inference from propositional knowledge bases interpreted under closed world assumptions has been investigated in [Eiter and Gottlob, 1993] [Cadoli and Lenzerini, 1994] [Coste-Marquis and Marquis, 1999]. Closed world reasoning typically is at the second level of the polynomial hierarchy. Especially, determining whether a clause is a logical consequence of a propositional formula interpreted under the extended closed world assumption is Π_2^p–complete (and this remains true when the clause is reduced to a literal and every atom of the formula is minimized). Some tractable restrictions also exist (see [Cadoli and Lenzerini, 1994] [Coste-Marquis and Marquis, 1999]for details).

4.4.2 Consequence finding characterization

For each form of closed world reasoning considered above, the corresponding free for negation formulas can be characterized through prime implicates.

Proposition 4.12 *Let Φ be a formula from $PROP_{PS}$ and $\langle P, Q, Z \rangle$ a partition of $Var(\Phi)$. Let Ψ be a formula from $PROP_{PS}$.*

- *Ψ is CWA-ffn given Φ iff Ψ is a positive literal and $\Psi \notin PI_{L^+_{PS}}(\Phi)$.*
- *Ψ is GCWA-ffn given Φ iff Ψ is a positive literal and $PI^{\Phi}_{L^+_{PS}}(\neg\Psi) = \emptyset$.*
- *Ψ is EGCWA-ffn given Φ iff Ψ is a conjunction of positive literals and $PI^{\Phi}_{L^+_{PS}}(\neg\Psi) = \emptyset$.*
- *Ψ is CCWA-ffn given Φ and $\langle P, Q, Z \rangle$ iff Ψ is a literal of L^+_P and $PI^{\Phi}_{L^+_P \cup L_Q}(\neg\Psi) = \emptyset$.*
- *Ψ is ECWA-ffn given Φ and $\langle P, Q, Z \rangle$ iff $Var(\Psi) \cap Z = \emptyset$ and $PI^{\Phi}_{L^+_P \cup L_Q}(\neg\Psi) = \emptyset$.*

Example 4.18 *Let Φ be the formula $((\neg a \wedge b) \Rightarrow c) \wedge b$. Let $P = \{a\}$, $Q = \{b\}$ and $Z = \{c\}$. We can check that:*

- *$PI_{L^+_{PS}}(\Phi) = \{b, (a \vee c)\}$. This set does not include a, c.*
- *$PI^{\Phi}_{L^+_{PS}}(\neg(a \wedge c)) = \emptyset$.*
- *$PI^{\Phi}_{L^+_P \cup L_Q}(\neg a) = \emptyset$.*

Many algorithms for computing circumscription take advantage of consequence finding techniques, especially those given in [Ginsberg, 1989] [Przymusinski, 1989] [Inoue, 1992]. In order to determine whether or not a formula Ψ can be derived from a completed knowledge base Φ, they search for explanations for Ψ (w.r.t. Φ) the negations of which are unexplainable. Intuitively, Ψ is a consequence of Φ interpreted under the extended closed world assumption iff there exists such an explanation. However, both the space of candidate formulas and the technique used to search this space differ with the algorithms.

Ginsberg's circumscriptive theorem prover is based on an abductive characterization of circumscription. Ginsberg shows that a formula Ψ is a consequence of a formula Φ interpreted under the extended closed world assumption w.r.t. $\langle P, \emptyset, Z \rangle$ iff there exists a disjunction γ of abductive explanations for Ψ w.r.t. $\langle \Phi, L^-_P \rangle$ s.t. $\neg\gamma$ is unexplainable. Interestingly, Ginsberg's result can be recovered as a specific case of Proposition 4.11 that gives an assumption-based characterization of inference from a formula interpreted under the extended closed world assumption and Proposition 4.9 that gives an abductive characterization of skeptical reasoning within an assumption-based theory. The key is that there exists a disjunction γ of abductive explanations for Ψ w.r.t. $\langle \Phi, L_P^- \cup L_Q \rangle$ s.t. $\neg\gamma$ is unexplainable iff the negation of the disjunction of all minimal abductive explanations for Ψ w.r.t. $\langle \Phi, L_P^- \cup L_Q \rangle$ is unexplainable (since this disjunction is equivalent to the logically weakest formula Σ s.t. $Lit(\Sigma) \subseteq L_P^- \cup L_Q$, $\Phi \wedge \Sigma$ is consistent and $\Phi \wedge \Sigma \models \Psi$ holds). Because the negation of this disjunction is equivalent to $PI^{\Phi}_{L_P^+ \cup L_Q}(\neg\Psi)$, the following characterization result can be derived:

Proposition 4.13 *Let Φ be a formula from $PROP_{PS}$ and $\langle P, Q, Z \rangle$ a partition of $Var(\Phi)$. Let Ψ be a formula from $PROP_{PS}$. $\Phi \vdash_{ECWA}^{\langle P,Q,Z \rangle} \Psi$ iff $PI^{\Phi}_{L_P^+ \cup L_Q}(\neg PI^{\Phi}_{L_P^+ \cup L_Q}(\neg\Psi)) = \emptyset$.*

Example 4.19 *Let Φ be the formula $((\neg a \wedge b) \Rightarrow c) \wedge b$. Let $P = \{a\}$, $Q = \{b\}$ and $Z = \{c\}$. We can check that $\Phi \vdash_{ECWA}^{\langle P,Q,Z \rangle} c$. Indeed, $PI^{\Phi}_{L_P^+ \cup L_Q}(\neg c) = \{a\}$, and $PI^{\Phi}_{L_P^+ \cup L_Q}(\neg a) = \emptyset$.*

In Ginsberg's approach, the formula $PI^{\Phi}_{L_P^+ \cup L_Q}(\neg\Psi)$ is computed using a specific ATMS based on multivalued logic. Such an ATMS is also used to compute $PI^{\Phi}_{L_P^+ \cup L_Q}(\neg PI^{\Phi}_{L_P^+ \cup L_Q}(\neg\Psi))$.

Contrastingly, Przymusinski uses *MILO-resolution*, a linear resolution strategy for computing circumscription[28]. MILO-resolution mainly is ordered linear resolution [Chang and Lee, 1973], enriched with a skipping rule (literals from $L_P^+ \cup L_Q$ can be skipped). Like in any linear resolution strategy, within MILO-resolution, the CNF formula Φ and an input clause Ψ (a top clause) are taken separately. MILO-resolution is oriented towards the generation of clauses containing no literal from $L_P^- \cup L_Z$ by resolving clauses from $\Phi \cup \{\Psi\}$ over such literals. Once such a clause has been generated as a central clause in a branch of the resolution tree, this branch is closed. In other words, literals from $L_P^+ \cup L_Q$ are systematically skipped. Thus, the MILO-resolution leaves are clauses without any literal from $L_P^- \cup L_Z$.

Definition 4.24 (MILO-resolution proof) *Let Φ be a CNF formula from $PROP_{PS}$ consisting of extended, reduced clauses and Ψ be a reduced clause from $PROP_{PS}$. Let γ be an extended, reduced clause. A* MILO-resolution proof *of γ from Φ and Ψ is a sequence γ_1, ..., γ_n of extended clauses s.t. $\gamma_1 = \Psi$, $\gamma_n = \gamma$, and every center clause γ_{i+1} ($i \in 1 .. n-1$) is generated from the previous center clause γ_i according to the following rules:*

[28] MILO-resolution stands for "MInimal model Linear Ordered resolution".

1. *An extended clause δ_{i+1} is constructed as the ordered resolvent of $\gamma_i = \langle l_1, ..., l_m\rangle$ and a side, input clause $\langle l'_1, ..., l'_r\rangle$ from Φ upon the first literal l_j of γ_i that belongs to $L^-_P \cup L_Z$. We have $\delta_{i+1} = \langle l_1, ..., l_{j-1}, l'_1, ..., l'_{t-1}, l'_{t+1}, ..., l'_r, [l_j], l_{j+1}, ..., l_m\rangle$, where $l'_t = \neg l_j$.*

2. *The clause γ_{i+1} is obtained from δ_{i+1} by performing the following reductions in the order specified:*

   (a) *deleting any unframed literal l in δ_{i+1} s.t. the framed literal $[\neg l]$ occurs in δ_{i+1};*

   (b) *merging any identical literals in δ_{i+1} from the left to the right;*

   (c) *removing any framed literals in δ_{i+1} that are not preceded by an unframed literal from $L^-_P \cup L_Z$.*

3. *γ_{i+1} is not a logical consequence of a previous center clause γ_j with $j < i$.*

The MILO-leaves *of Φ and top clause Ψ are the clauses γ that do not contain any literal from $L^-_P \cup L_Z$ and have a MILO-resolution proof from Φ and Ψ.*

Example 4.20 *Let Φ be the CNF formula consisting of the extended clauses $\langle a, \neg b, c\rangle$ and $\langle c\rangle$. Let $P = \{a\}$, $Q = \{b\}$ and $Z = \{c\}$. The following sequence is a MILO-resolution proof of $\langle a, \neg b, [\neg c]\rangle$ from Φ and top clause $\neg c$:*

1. $\langle \neg c\rangle$

2. $\langle a, \neg b, [\neg c]\rangle$

The produced extended clause $\langle a, \neg b, [\neg c]\rangle$ of this proof is nothing but a representation of the clause $(a \vee \neg b)$ (the reader should remember that framed literals is just a way to keep track of the previous literals resolved upon within each central clause, so as to avoid performing useless resolutions). This produced clause is a MILO-leaf of Φ and $\neg c$ (since it does not contain a literal from $L^-_P \cup L_Z$, it cannot be resolved further).

Given a top clause Ψ (the root of the derivation tree), every leaf clause is a $L^+_P \cup L_Q$-implicate of $\Phi \wedge \Psi$. Moreover, no such implicate can be missed up to Φ-equivalence in the sense that every $L^+_P \cup L_Q$-implicate of $\Phi \wedge \Psi$ is necessarily a Φ-consequence of the conjunction of all leaves of the MILO-resolution tree with Ψ as top clause. Accordingly, the conjunction of all leaves of such a tree is Φ-equivalent to the conjunction of every $L^+_P \cup L_Q$-implicate of $\Phi \wedge \Psi$ (hence Φ-equivalent to $PI_{L^+_P \cup L_Q}(\Phi \wedge \Psi)$).

Example 4.21 *Let Φ be the CNF formula consisting of the extended clauses $\langle a, \neg b, c\rangle$ and $\langle c\rangle$. Let $P = \{a\}$, $Q = \{b\}$ and $Z = \{c\}$. $(a \vee \neg b)$ is the unique MILO-leaf of Φ and $\neg c$. We can check that $(a \vee \neg b)$ is Φ-equivalent to $PI_{L^+_P \cup L_Q}(\Phi \wedge \neg c) = \{(a \vee \neg b)\}$.*

So as to check whether a given clause Ψ is a consequence of a CNF formula Φ interpreted under the extended closed world assumption w.r.t. $\langle P, Q, Z\rangle$, Przymusinski considers two situations: Ψ does not contain any literal from L_Z, and the general case where no restriction over Ψ is considered. Indeed, in the first case, $\Phi \vdash_{ECWA}^{\langle P,Q,Z\rangle} \Psi$ holds iff $\neg\Psi$ is ECWA-ffn given Φ and $\langle P, Q, Z\rangle$. Accordingly, it is sufficient to check whether or not $PI^{\Phi}_{L_P^+ \cup L_Q}(\Psi) = \emptyset$ holds, or equivalently, to check whether or not $\Phi \models PI_{L_P^+ \cup L_Q}(\Phi \wedge \Psi)$ holds. Since $PI_{L_P^+ \cup L_Q}(\Phi \wedge \Psi)$ is Φ-equivalent to the conjunction of every MILO-leaf clause of Φ and Ψ as a top clause, it is sufficient to find out one such leaf clause that is not a logical consequence of Φ to conclude that Ψ is not a consequence of the completed knowledge base Φ (and the converse also holds). Now, in the case where Ψ contains some literals from Z, the approach by Przymusinski first consists in eliminating them thanks to the next proposition:

Proposition 4.14 *Let Φ be a formula from $PROP_{PS}$ and $\langle P, Q, Z\rangle$ a partition of $Var(\Phi)$. Let Ψ be a formula from $PROP_{PS}$. $\Phi \vdash_{ECWA}^{\langle P,Q,Z\rangle} \Psi$ iff $\Phi \vdash_{ECWA}^{\langle P,Q,Z\rangle} \neg PI_{L_P^+ \cup L_Q}(\Phi \wedge \neg\Psi)$.*

Example 4.22 *Let Φ be the CNF formula consisting of the extended clauses $\langle a, \neg b, c\rangle$ and $\langle c\rangle$. Let $P = \{a\}$, $Q = \{b\}$ and $Z = \{c\}$. We can check that $\Phi \vdash_{ECWA}^{\langle P,Q,Z\rangle} c$ holds. To do it, we verify that $\neg PI_{L_P^+ \cup L_Q}(\Phi \wedge \neg c)$ is a consequence of Φ interpreted under the extended closed world assumption w.r.t. $\langle P, Q, Z\rangle$. We have $\neg PI_{L_P^+ \cup L_Q}(\Phi \wedge \neg c) = (\neg a \wedge b)$. So it remains to check that both $\neg a$ (resp. b) is inferable, which can be done by observing that there is no MILO-leaf of Φ and $\neg a$ (resp. b).*

Since $\neg PI_{L_P^+ \cup L_Q}(\Phi \wedge \neg\Psi)$ is a formula that does not contain any literal from L_Z, it remains to check whether every clause from a CNF of this formula is a consequence of Φ interpreted under the extended closed world assumption w.r.t. $\langle P, Q, Z\rangle$ (the general situation has been reduced to the first situation). Actually, a formula Φ-equivalent to $PI_{L_P^+ \cup L_Q}(\Phi \wedge \neg\Psi)$ is computed using MILO-resolution: since the negations of the m literals contained in $\Psi = l_1 \vee ... \vee l_m$ cannot be considered simultaneously as top clauses, they are considered successively (m MILO-resolution trees are computed and for the i^{th} one, the top clause is $\neg l_i$ and the corresponding side clauses are $\Phi \cup \{\neg l_1, ..., \neg l_{i-1}\}$). The conjunction of all the leaf clauses of these m trees is the derived formula.

Example 4.23 *Let Φ be the CNF formula consisting of the extended clauses $\langle a, \neg b, c\rangle$ and $\langle c\rangle$. Let $P = \{a\}$, $Q = \{b\}$ and $Z = \{c\}$. We can check that $\Phi \vdash_{ECWA}^{\langle P,Q,Z\rangle} \Psi$ holds by verifying that the negation of the conjunction of all MILO-leaves of Φ and $\neg c$ satisfies the same property. This conjunction is* *reduced to the clause $(a \vee \neg b)$, and $\neg a$ and b are both inferable (see the example above).*

Like Ginsberg's, Przymusinski's approach can be considered as based on Proposition 4.13. On the one hand, instead of computing $PI^{\Phi}_{L_P^+ \cup L_Q}(\neg\Psi)$, a Φ-equivalent formula Σ is computed (and this has no impact on the correctness of the proposition since Φ-equivalent formulas share the same $\langle L_P^+ \cup L_Q, \Phi\rangle$-prime implicates. On the other hand, instead of checking whether the computed formula Σ is unexplainable by testing the vacuity of the set of $\langle L_P^+ \cup L_Q, \Phi\rangle$-prime implicates of its negation, we check whether or not a formula Φ-equivalent to the set of $L_P^+ \cup L_Q$-prime implicates of $\Phi \wedge \neg\Sigma$ is a logical consequence of Φ.

Each algorithm has some features that makes it computationally more efficient than the other one in some situations [Inoue and Helft, 1990]. On the one hand, MILO-resolution is oriented towards the generation of clauses containing no literal from $L_P^- \cup L_Z$, hence many computational efforts that would be spent, would a general theorem prover be used, are avoided. Especially, whenever a clause Ψ containing no literal from $L_P^- \cup L_Z$ is reached, the corresponding branch of the resolution tree is closed. All the clauses containing no literal from $L_P^- \cup L_Z$ that could be obtained by resolving Ψ with clauses of Φ are logical consequences of $\Phi \wedge \Psi$ or, in other words, they are Φ-consequences of Ψ. Hence, when Ψ has been computed, deriving these clauses is useless. Such clauses can be generated by Ginsberg's algorithm. On the other hand, within Ginsberg's algorithm, the clauses that are generated never are logical consequences of Φ (since they are $\langle L_P^+ \cup L_Q, \Phi\rangle$-prime implicates). This contrasts with Przymusinski's approach where the clauses generated by MILO-resolution can easily be entailed by Φ. All such clauses are obviously unnecessary since, as discussed above, the computed formulas can be considered up to Φ-equivalence.

Inoue [Inoue, 1992] shows how Przymusinski's approach can be improved by using a variant of SOL-resolution (namely SOL_S-resolution, the one in which the skipping rule is applied in priority and prevents the resolution rule from being fired each time skipping a literal is possible). Computational improvements come from the fact that SOL_S-resolution is based on model elimination, which is a version of linear resolution that is superior to ordered linear resolution (on which MILO-resolution is based). In contrast to SOL-resolution, completeness in characteristic consequence finding is not preserved by SOL_S-resolution. However, every $L_P^+ \cup L_Q$-implicate of $\Phi \wedge \Psi$ is necessarily a Φ-consequence of the conjunction of all leaves of a SOL_S-resolution tree with Ψ as top clause. As explained above, this is sufficient for the purpose of computing circumscription.

While the techniques above show how consequence finding algorithms can be used so as to query a knowledge base interpreted under the extended closed world assumption, consequence finding can also be used to characterize the completed knowledge base itself, i.e. independently from any query. Indeed, a knowledge base interpreted under the extended closed world assumption can be characterized directly from prime implicates [Raiman and de Kleer, 1992]:

Proposition 4.15 *Let Φ be a formula from $PROP_{PS}$ and $\langle P, Q, Z\rangle$ a partition of $Var(\Phi)$, where $P = \{p_1, ..., p_n\}$. Let Ψ be a formula from $PROP_{PS}$. $\Phi \vdash_{ECWA}^{\langle P,Q,Z\rangle} \Psi$ iff Ψ is a logical consequence of the formula $\Phi \wedge (PI^{\Phi}_{L_P^+ \cup L_Q}(\neg p_1) \Rightarrow \neg p_1) \wedge ... \wedge (PI^{\Phi}_{L_P^+ \cup L_Q}(\neg p_n) \Rightarrow \neg p_n)$.*

Example 4.24 *Let Φ be the formula $((\neg a \wedge b) \Rightarrow c) \wedge b$. Let $P = \{a\}$, $Q = \{b\}$ and $Z = \{c\}$. We have $PI^{\Phi}_{L_P^+ \cup L_Q}(\neg a) = \emptyset$. Hence Φ interpreted under the extended closed world assumption w.r.t. $\langle P, Q, Z\rangle$ is equivalent to $\Phi \wedge \neg a$.*

Note that such an approach has a compilation flavor [Cadoli and Donini, 1998] in the sense that the completed knowledge base is turned into a classical formula and inference from such a formula is easier than inference from a completed knowledge base, unless the polynomial hierarchy collapses (deduction from a propositional formula typically is coNP-complete, while deduction from a formula interpreted under the extended closed world assumption is Π_2^p–complete in the general case). Provided that the size of the compilation remains polynomial in the size of the original knowledge base (which is not guaranteed here, and is very unlikely whatever the compilation technique [Cadoli *et al.*, 1997]), the compilation time can be balanced once sufficiently many queries have been considered.

4.5 Clause management systems

Clause management systems (CMSs) have been introduced in [Reiter and de Kleer, 1987] as a generalization of assumption-based truth maintenance systems (ATMSs) [de Kleer, 1986]. Additional formal developments and various applications have been proposed in [Tsiknis and Kean, 1988] [Kean and Tsiknis, 1990a] [Kean, 1992] [Kean and Tsiknis, 1992] [Kean and Tsiknis, 1993]. An extension of CMSs to circumscribed knowledge bases (called MMS, minimality maintenance system) has been proposed in [Raiman and de Kleer, 1992]. All these systems enable implementations of various forms of assumption-based reasoning to be achieved, especially abduction. Technically, a CMS implements consequence finding through the derivation of prime implicates.

4.5.1 Definitions

From a computational point of view, many forms of reasoning in artificial intelligence can be considered as search through a solution space. For many problems, the search space is so huge that there is a need for supplementary systems that can aid the reasoning mechanisms to circumvent this difficulty by exploiting as much as possible the information that have been collected through the search. Especially, chronological backtracking must be avoided when a more intelligent form of backtracking is possible.

A *clause management system* (CMS for short) is a system that can help a reasoner by keeping track of its inferences and reasoning about them. Coupled with a reasoner, it receives from it some information, typically under the form of propositional clauses. Such clauses are abstract representations of the reasoner activity. For instance, if the reasoner has been able to derive a clause γ from the set Ψ of atoms, then it can decide to transmit the clause corresponding to the implicative formula $\Psi \Rightarrow \gamma$ (which represents the trace of this inference) to the CMS. As a counterpart, the reasoner can ask the CMS for explanations of some clauses γ, i.e., the most general contexts Ψ in which γ is known to hold, given the knowledge base. Thus, in any context Ψ, the reasoner can easily know whether a clause γ has already been shown true, without any additional inference: it just has to ask the CMS for it. Accordingly, a CMS can be viewed as a domain-independent intelligent cache.

The key notion CMSs rely on is the notion of *minimal support* for a clause.

Definition 4.25 (support, minimal support) *Let Φ be a CNF formula from $PROP_{PS}$ and γ be a clause of $PROP_{PS}$.*

- *A* support *clause for γ w.r.t. Φ is a clause π from $PROP_{PS}/\equiv$ s.t. $\Phi \models (\pi \vee \gamma)$ and $\Phi \not\models \pi$ hold.*
- *A* minimal support *clause for γ w.r.t. Φ is a support clause for γ w.r.t. Φ which is minimal w.r.t. $\models$ (up to logical equivalence).*

Example 4.25 *Let $\Phi = \{(a \Rightarrow b)\}$ and $\gamma = b$. The minimal support clauses for γ w.r.t. Φ are $\neg b$ (a trivial support) and $\neg a$ (up to logical equivalence).*

In the definition above, Φ represents the set of propositional clauses that have been transmitted so far by the reasoner to the CMS. The purpose of the CMS is to compute the set of minimal supports for clause γ each time the reasoner asks for it.

Note that in the original definition of a support clause [Reiter and de Kleer, 1987], a more syntactical view is given to both notions of support and minimal support. Especially, minimality is considered w.r.t. set-inclusion (instead of logical entailment). We nevertheless adhere to the definition above since it proves more adequate in some situations.

For many applications (especially those related to assumption-based reasoning), we are not interested in any (minimal) support, but only on those that are built up from a given set L of literals. So as to take this constraint into account, Kean and Tsiknis [Kean and Tsiknis, 1990a] [Kean and Tsiknis, 1992] extend basic CMS to so-called *assumption-based clause management systems* (ACMSs for short). The purpose of ACMS basically is to compute L-minimal supports.

Definition 4.26 (L-support, L-minimal support) *Let Φ be a CNF formula from $PROP_{PS}$ and γ be a clause of $PROP_{PS}$. Let L be a finite subset of L_{PS}.*

- *A L-*support *clause for γ w.r.t. Φ is a clause π from $PROP_{PS}/\equiv$ s.t. $Lit(\pi) \subseteq L$, $\Phi \models (\pi \vee \gamma)$ and $\Phi \not\models \pi$ hold.*
- *A L-*minimal support *clause for γ w.r.t. Φ is a L-support clause for γ w.r.t. Φ which is minimal w.r.t. $\models$ (up to logical equivalence).*

Example 4.26 *Let $\Phi = \{(a \Rightarrow b)\}$, $\gamma = b$, $L = \{\neg a\}$. $\neg a$ is the unique L-minimal support for γ w.r.t. Φ (up to logical equivalence). If $L = \{b\}$, then γ has no L-support w.r.t. Φ.*

As shown in [Reiter and de Kleer, 1987], de Kleer's ATMS [de Kleer, 1986] can be considered as a restricted ACMS the purpose of which is to compute the set of all L-minimal supports for γ where L is a given set of atoms (positive literals), Φ is a Horn formula, and γ is a positive literal. In ATMS terminology, the negations of the L-minimal support clauses for γ w.r.t. Φ form a consistent, sound, complete and minimal label for γ.

4.5.2 Consequence finding characterization

As the previous example illustrates it, the minimal supports (resp. L-minimal supports) for γ are just the negations of the minimal abductive explanations for it, where the set of individual hypotheses is L_{PS} (resp. L). Accordingly, minimal supports (resp. L-minimal supports) can be characterized as Φ-prime implicates (resp. $\langle L, \Phi\rangle$-prime implicates):

Proposition 4.16 *Let Φ be a CNF formula from $PROP_{PS}$ and γ be a clause of $PROP_{PS}$. The set of all minimal support clauses for γ w.r.t. Φ is $PI^{\Phi}(\neg\gamma)$ (up to logical equivalence).*

Proposition 4.17 *Let Φ be a CNF formula from $PROP_{PS}$, γ be a clause of $PROP_{PS}$ and L be a finite subset of L_{PS}. The set of all L-minimal support clauses for γ w.r.t. Φ is $PI^{\Phi}_{L}(\neg\gamma)$ (up to logical equivalence).*

Conversely, the Φ-prime implicates (resp. $\langle L, \Phi\rangle$-prime implicates) of a formula can be easily characterized through its minimal supports (resp. L-minimal supports). Actually, the notions are very close and each of them can be considered as an alternative to the other. This connection is important for several reasons. On the one hand, it shows that CMS/ACMS can be extended so as to compute supports for any formula (and not only for clauses as in its current turn). Accordingly, Φ must not be necessarily restricted to a CNF formula but any propositional formula could be considered. On the other hand, while CMS basically use (variants of) Proposition 3.21 for computing supports, all the computational techniques pointed out in this chapter for computing prime implicates can be adapted to derive supports. As an example, Inoue [Inoue, 1990] [Inoue, 1992] shows how SOL-resolution can be used so as to derive minimal supports and L-minimal supports. Clearly enough, since many kinds of common sense inference can be characterized through some notions of prime implicate, they can also be characterized through the corresponding notions of support. Thus, ACMS furnishes many basic computational tools for implementing forms of common sense reasoning and has been used to this end in several implementations.

5 Conclusion

This chapter is centered on consequence finding. Many computational techniques have been presented and the role played by consequence finding for characterizing several forms of common sense reasoning, including abduction, simple default reasoning, circumscription and closed world reasoning has been emphasized as well.

Consequence finding algorithms typically consist in computing some kinds of prime implicates, or formulas related to such implicates. Many techniques have been pointed out in this chapter for finding out some characteristic theorems.

The proposed approaches are however of a limited computational scope (algorithms do not scale up well) and there is only little hope that significantly better algorithms could be designed. Indeed, from a theoretical point of view, consequence finding has been shown hard. Roughly, there are two sources of complexity: the number of theorems of interest (which can be exponential in the input size), and the difficulty to recognize them (the corresponding decision problems are typically at the first or the second level of the polynomial hierarchy). These two sources of complexity are still encountered when consequence finding is used for common sense reasoning (e.g., there can be exponentially many free for negation formulas w.r.t. some forms of circumscription, checking whether an abductive explanation exists is hard, etc.).

From a practical point of view, it is difficult to derive firm conclusions about the relative efficiency of existing prime implicates algorithms. Only few empirical comparisons have been performed so far, especially for the sophisticated forms of prime implicates, like L-prime implicates or $\langle L, \Phi \rangle$-

prime implicates. Thus, despite their goal-directedness and low memory usage, the performance of linear resolution strategies (like SOL-resolution) for consequence finding is still controversial [del Val, 1999]. In our opinion, there is no best algorithm for generating prime implicates, i.e., one that would be the most efficient w.r.t. time complexity or space complexity. Actually, the performances of existing algorithms mainly depend on the nature of the input, e.g., the fact that the knowledge base is inconsistent or has only few prime implicates or that the set of literals we focus on is small or that the knowledge base corresponds to a monotone boolean function... What renders a comparison more difficult is that existing algorithms often assume a specific representation of the knowledge base; the input is required to be encoded as a CNF formula, a DNF formula, a NNF formula, a conjunction of DNF formulas, a BDD, the whole set of its canonical minterms [29], etc. Moreover, some algorithms have been specifically designed for the incremental handling of prime implicates, i.e., part of the input is given by its prime implicates and the purpose is to compute the prime implicates of an extended knowledge base. Thus, in practice, it is important to take advantage of the properties of the specific instances that are considered for choosing the "right" computational technique.

Consequence finding algorithms prove helpful for common sense reasoning by reducing it to assumption-based reasoning: inference is considered as deduction from a knowledge base completed with some formulas that have consequence finding characterizations. One of the advantages of the assumption-based approach to common sense reasoning is that it makes inference deductive inside a standard monotonic logical framework. And deduction in this framework is well-known and understood, compared with the other formalisms developed so far for various forms of common sense inference. Moreover, making the assumptions explicit can help controlling inference: in contrast to other forms of common sense reasoning, inference from a completed knowledge base can be viewed as a two-steps process: some assumptions are elected, then deductive reasoning from them and the prior knowledge is conducted; thus, assumption-based reasoning gives a way to look inside common sense inference in some sense: one step of inference is decomposed into two steps.

Of course, there is a price to be paid for the gain in intelligibility. Indeed, an intensional characterization of the additional knowledge under which inference must be drawn is often much more concise than an explicit representation of it. While making assumptions explicit can be a good idea from a logical point of view (rendering the contents of the knowledge base interpreted in a non-standard way more easy to grasp), this is not always

[29] More "exotic" representations can also be found, see e.g., [Pizzuti, 1996] [Palopoli *et al.*, 1999] where the input formula is represented as a set of constraints in a integer programming setting.

possible and, when possible, this is not always valuable from a computational point of view. Thus, in the first-order case, the additional knowledge represented by completion policies cannot always be explicited as formulas from the same language as the one of the original knowledge base. For instance, domain closure assumptions (stating roughly that the domain of any variable is reduced to a given set) cannot always be represented as a first-order formula, especially when the corresponding domain is not finite - since a non-finite disjunction is not a formula. In the same vein, while there exist classes of first-order formulas the circumscription of which is equivalent to a first-order formula (e.g., separable formulas [Lifschitz, 1985] and collapsible formulas [Rabinov, 1989] when there are no varying predicates, existentially quantified formulas [Kolaitis and Papadimitriou, 1988] [Kolaitis and Papadimitriou, 1990], or the more general class given in [Doherty *et al.*, 1995]), such classes are not always recursive [Krishnaprasad, 1988] and in, the general case, the circumscription of a first-order formula is a second-order (universal) formula which is not equivalent to some first-order one [Krishnaprasad, 1988] [Kolaitis and Papadimitriou, 1990]. Accordingly, assumption-based reasoning does not cover all the forms of logic-based non-monotonic inference.

In the propositional case, there are situations for which reducing common sense reasoning to standard deduction can prove computationally helpful. Especially, when a propositional knowledge base interpreted under some implicit assumptions can be compiled into a propositional formula interpreted in a classical way, some computational benefits can be achieved sometimes since common sense propositional reasoning usually is at the second level of the polynomial hierarchy while deductive propositional inference is at the first level, only. To be more precise, as long as the compilation time can be balanced by many queries and the size of the compilation remains polynomial in the size of the original knowledge base, compilation can prove computationally valuable. For instance, Ben-Eliyahu and Dechter [Ben-Eliyahu and Dechter, 1996] show how some restricted propositional default theories can be compiled into propositional formulas in polynomial time (hence, polynomial space too). Subsequently, inference from such theories proves easier than inference from general propositional default theories (unless the polynomial hierarchy collapses). Unfortunately, making the assumptions explicit does not prove computationally valuable in the general case. Indeed, Cadoli and his colleagues [Cadoli *et al.*, 1997] show that the possibility to translate the circumscription of a propositional formula Σ into an equivalent propositional formula of size polynomial in $|\Sigma|$ would imply the polynomial hierarchy to collapse; stated otherwise, the size of the compilation of some Σ (interpreted under circumscription) is likely to be exponential in the size of Σ, rendering the computational benefit of compilation hard to be believed in the worst case. Several related results for restricted cases of circumscription and default logic can also be found in [Cadoli *et al.*, 1996] and [Gogic *et al.*, 1995].

While the notion of consequence finding is central for many AI applications, its current scope can nevertheless be considered limited, so subject to extension in at least two directions.

On the one hand, consequence finding could be used to characterize some additional forms of common sense inference. In particular, inference in so-called propositional X-logic [Siegel and Forget, 1996], defined by $\Phi \vdash_X \Psi$ holds iff every logical consequence of $\Phi \wedge \Psi$ that belongs to the set of formulas X also is a logical consequence of Φ, has a consequence finding characterization whenever X is the set of all formulas that are built up from a given set L of literals. Indeed, in this situation, it is easy to show that $\Phi \vdash_X \Psi$ holds iff $PI_L^{\Phi}(\Psi) = \emptyset$. Extending this result to the case where X is not deductively closed is an issue for further research.

On the other hand, consequence finding is based on classical deduction in the sense that the theorems that are looked for are logical consequences of the given knowledge base. However, in many situations, interesting theorems are not limited to such standard consequences but include formulas that are derivable through some forms of common sense inference that are merely not reducible to deduction. To deal with such cases, an interesting perspective would be to extend consequence finding so as to handle more general forms of theorems, and use it for characterizing sophisticated forms of common sense inference. Some researches can be viewed as a first step in this direction. Thus, in [Raiman and de Kleer, 1992], abduction is defined from a circumscribed theory. In [Eiter *et al.*, 1995], abduction is defined from a default theory. Obviously, extending the definition of prime implicates considered in this chapter to a more general framework is an easy task (just replace $\models$ by $\vdash$, where $\vdash$ is any inference relation). However, many works remain to be done in the study of such prime implicates and the design of dedicated computational techniques.

Acknowledgements

I would like to thank Alvaro del Val, Hélène Fargier, Andreas Herzig, Jérôme Lang and Daniel Le Berre for a careful reading of a preliminary draft of this chapter. Of course, all the remaining flaws are of my own. This work has been supported by a "Contrat d'objectifs de la Région Nord/Pas–de–Calais" and by the IUT de Lens.

References

[Akers, 1978] S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.

[Ben-Eliyahu and Dechter, 1996] R. Ben-Eliyahu and R. Dechter. Default reasoning using classical logic. *Artificial Intelligence*, 84:113–150, 1996.

[Besnard and Siegel, 1988] Ph. Besnard and P. Siegel. *The preferential-models approach to non-monotonic logics*, chapter Non-Standard Logics for Automated Reasoning, pages 137–161. Academic Press, London, 1988.

[Besnard, 1989] Ph. Besnard. *Default logic.* Symbolic Computation. Springer-Verlag, Berlin, 1989.

[Bibel, 1981] W. Bibel. On matrices with connections. *Journal of the Association for Computing Machinery*, 28(4):633–645, 1981.

[Bibel, 1982] W. Bibel. A comparative study of several proof procedures. *Artificial Intelligence*, 18:269–293, 1982.

[Blake, 1937] A. Blake. *Canonical expressions in boolean algebra.* PhD thesis, University of Chicago, Chicago (IL), 1937.

[Boï *et al.*, 1992] J.-M. Boï, E. Innocente, A. Rauzy, and P. Siegel. Production fields: a new approach to deduction problems and two algorithms for propositional calculus. *Revue d'intelligence artificielle*, 6(3):235–253, 1992.

[Boros *et al.*, 1990] E. Boros, Y. Crama, and P.L. Hammer. Polynomial-time inference of all valid implications for Horn and related formulae. *Annals of Mathematics and Artificial Intelligence*, 1:21–32, 1990.

[Bossu and Siegel, 1985] G. Bossu and P. Siegel. Saturation, nonmonotonic reasoning and the closed-world assumption. *Artificial Intelligence*, 25:23–67, 1985.

[Brewka, 1989] G. Brewka. Preferred subtheories: an extended logical framework for default reasoning. In *Proc. of the* 11^{th} *International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1043–1048, Detroit (MI), 1989.

[Bryant, 1986] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[Cadoli and Donini, 1998] M. Cadoli and F.M. Donini. A survey on knowledge compilation. *AI Communications*, 1998. (in press).

[Cadoli and Lenzerini, 1994] M. Cadoli and M. Lenzerini. The complexity of propositional closed world reasoning and circumscription. *Journal of Computer and System Sciences*, 48:255–310, 1994.

[Cadoli *et al.*, 1992] M. Cadoli, Th. Eiter, and G. Gottlob. An efficient method for eliminating varying predicates from a circumscription. *Artificial Intelligence*, 54:397–410, 1992.

[Cadoli *et al.*, 1996] M. Cadoli, F.M. Donini, and M. Schaerf. Is intractability of non-monotonic reasoning a real drawback? *Artificial Intelligence*, 88:215–251, 1996.

[Cadoli *et al.*, 1997] M. Cadoli, F.M. Donini, M. Schaerf, and R. Silvestri. On compact representations of propositional circumscription. *Theoretical Computer Science*, 182:183–202, 1997.

[Castell, 1996] Th. Castell. Computation of prime implicates and prime implicants by a variant of the Davis and Putnam procedure. In *Proc. of the* 8^{th} *IEEE International Conference on Tools with Artificial Intelligence (ICTAI'96)*, pages 428–429, Toulouse, 1996.

[Castell, 1997] Th. Castell. *Consistance et déduction en logique propositionnelle.* Thèse d'université, Université Paul Sabatier, Toulouse, 1997. (in french).

[Cayrol, 1992] C. Cayrol. Une modèle logique général pour le raisonnement révisable. *Revue d'Intelligence Artificielle*, 6(3), 1992.

[Cazalens, 1992] S. Cazalens. *Formalisation en logique non standard de certaines méthodes de raisonnement pour fournir des réponses coopératives, dans des systèmes de bases de donneés et de connaissances.* Thèse d'université, Université Paul Sabatier, Toulouse, 1992. (in french).

[Chandra and Markowsky, 1978] A.K. Chandra and G. Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24:7–11, 1978.

[Chang and Lee, 1973] C.L. Chang and R.C.T. Lee. *Symbolic logic and mechanical theorem proving.* Academic Press, New York, 1973.

[Chang, 1970] C.L. Chang. The unit proof and the input proof in theorem proving. *Journal of the Association for Computing Machinery*, 17(4):698–707, 1970.

[Console *et al.*, 1989] L. Console, D. Theseider Dupré, and P. Torasso. A theory of diagnosis for incomplete causal models. In *Proc. of the* 11^{th} *International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1131–1137, Detroit (MI), 1989.

[Console *et al.*, 1991] L. Console, D. Theseider Dupré, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1:661–690, 1991.

[Coste-Marquis and Marquis, 1999] S. Coste-Marquis and P. Marquis. Complexity results for propositional closed world reasoning and circumscription from tractable knowledge bases. In *Proc. of the* 16^{th} *International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 24–29, Stockholm, 1999.

[Cox and Pietrzykowski, 1986] P.T. Cox and T. Pietrzykowski. Causes for events: their computation and applications. In *Proc. of the* 8^{th} *International Conference on Automated Deduction (CADE'86)*, pages 608–621, Oxford, 1986.

[Cox and Pietrzykowski, 1987] P.T. Cox and T. Pietrzykowski. General diagnosis by abductive inference. In *Proc. of the IEEE Symposium on Logic Programming*, pages 183–189, 1987.

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the Association for Computing Machinery*, 5:394–397, 1962.

[de Kleer and Konolige, 1989] J. de Kleer and K. Konolige. Eliminating the fixed predicates from a circumscription. *Artificial Intelligence*, 39:391–398, 1989.

[de Kleer, 1986] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–167, 1986.

[de Kleer, 1992] J. de Kleer. An improved incremental algorithm for generating prime implicates. In *Proc. of the* 10^{th} *National Conference on Artificial Intelligence (AAAI'92)*, pages 780–785, San Jose (CA), 1992.

[Dechter and Rish, 1994] R. Dechter and I. Rish. Directional resolution: the Davis-Putnam procedure, revisited. In *Proc. of the* 4^{th} *International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 134–145, Bonn, 1994.

[del Val, 1994] A. del Val. Tractable databases: how to make propositional unit resolution complete through compilation. In *Proc. of the* 4^{th} *International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 551–561, Bonn, 1994.

[del Val, 1995] A. del Val. An analysis of approximate knowledge compilation. In *Proc. of the* 14^{th} *International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 830–836, Montreal, 1995.

[del Val, 1996] A. del Val. Approximate knowledge compilation: the first-order case. In *Proc. of the* 13^{th} *National Conference on Artificial Intelligence (AAAI'96)*, pages 498–503, Portland (OR), 1996.

[del Val, 1999] A. del Val. A new method for consequence finding and compilation in restricted language. In *Proc. of the* 16^{th} *National Conference on Artificial Intelligence (AAAI'99)*, pages 259–264, Orlando (FL), 1999.

[Demolombe and del Cerro, 1991] R. Demolombe and L. Fariñas del Cerro. An inference rule for hypothesis generation. In *Proc. of the* 12^{th} *International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 152–157, Sydney, 1991.

[Doherty *et al.*, 1995] P. Doherty, W. Lukaszewicz, and A. Szalas. Computing circumscription revisited: preliminary report. In *Proc. of the* 14^{th} *International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1502–1508, Montreal, 1995.

[Dunham and Fridshal, 1959] B. Dunham and B. Fridshal. The problem of simplifying logical expressions. *Journal of Symbolic Logic*, 1959.

[Eiter and Gottlob, 1993] Th. Eiter and G. Gottlob. Propositional circumscription and extended closed-world reasoning are Π_2^p-complete. *Theoretical Computer Science*, 114:231–245, 1993.

[Eiter and Gottlob, 1995] Th. Eiter and G. Gottlob. The complexity of logic-based abduction. *Journal of the Association for Computing Machinery*, 42(1):3–42, 1995.

[Eiter *et al.*, 1995] Th. Eiter, G. Gottlob, and N. Leone. Semantics and complexity of abduction from default theories. In *Proc. of the* 14^{th} *International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 870–876, Montreal, 1995.

[Eshghi, 1993] K. Eshghi. A tractable class of abduction problems. In *Proc. of the* 13^{th} *International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 3–8, Chambéry, 1993.

[Etherington, 1987] D.V. Etherington. Relating default logic and circumscription. In *Proc. of the* 10^{th} *International Joint Conference on Artificial Intelligence (IJCAI'87)*, pages 489–494, Milano, 1987.

[Finger and Genesereth, 1985] JJ. Finger and M.R. Genesereth. Residue: a deductive approach to design synthesis. Technical Report STAN-CS-85-1035, Department of Computer Science, Stanford University, Stanford (CA), 1985.

[Finger, 1987] J.J. Finger. *Exploiting constraints in design synthesis*. PhD thesis, Stanford University, Stanford, 1987.

[Fridshal, 1957] R. Fridshal. The Quine algorithm. In *Summaries of talks at the summer institute of symbolic logic*, page 211, Cornell University, 1957.

[Friedman and Supowit, 1990] S.J. Friedman and K.J. Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers*, C-39:710–713, 1990.

[Gelfond and Przymusinska, 1986] M. Gelfond and H. Przymusinska. Negation as failure: careful closure procedure. *Artificial Intelligence*, 30:273–287, 1986.

[Gelfond *et al.*, 1986] M. Gelfond, H. Przymusinska, and T. Przymusinski. The extended closed world assumption and its relationship to parallel circumscription. In *Proc. of ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS'86)*, pages 133–139, 1986.

[Gelfond *et al.*, 1989] M. Gelfond, H. Przymusinska, and T. Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38:49–73, 1989.

[Ginsberg, 1989] M. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:109–230, 1989.

[Gogic *et al.*, 1995] G. Gogic, H.A. Kautz, Ch.H. Papadimitriou, and B. Selman. The comparative linguistics of knowledge representation. In *Proc. of the* 14^{th} *International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 862–869, Montreal, 1995.

[Gottlob, 1992] G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2:397–425, 1992.

[Hobbs *et al.*, 1993] J. Hobbs, M.E. Stickel, D. Appelt, and P. Martin. Interpretation as abduction. *Artificial Intelligence*, 63:69–142, 1993.

[Hwa, 1974] H.R. Hwa. A method for generating prime implicants of a boolean expression. *IEEE Transactions in Computers*, pages 637–641, 1974.

[Igarashi, 1979] Y. Igarashi. An improved lower boound on the maximum number of prime implicants. *Transaction of the IECE of Japan*, E-62(6):389–394, 1979.

[Inoue and Helft, 1990] K. Inoue and N. Helft. On theorem provers for circumscription. In *Proc. of the* 8^{th} *Biennal Conference of the Canadian*

Society for Computational Studies of Intelligence (CSCSI'90), pages 212–219, Ottawa, 1990.

[Inoue, 1990] K. Inoue. An bductive procedure for the CMS/ATMS. In *Proc. of the ECAI'90 Workshop on Truth Maintenance Systems*, volume 515 of *Lecture Notes in Artificial Intelligence*, pages 34–53, Stockholm, 1990. Springer–Verlag.

[Inoue, 1991] K. Inoue. Consequence-finding based on ordered linear resolution. In *Proc. of the* 12^{th} *International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 158–164, Sydney, 1991.

[Inoue, 1992] K. Inoue. Linear resolution in consequence–finding. *Artificial Intelligence*, 56(2–3):301–353, 1992.

[Jackson and Pais, 1990] P. Jackson and J. Pais. Computing prime implicants. In *Proc. of the* 10^{th} *International Conference on Automated Deduction (CADE'90)*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 543–557, Kaiserslautern, 1990. Springer–Verlag.

[Jackson, 1992] P. Jackson. Computing prime implicants incrementally. In *Proc. of the* 11^{th} *International Conference on Automated Deduction (CADE'92)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 253–267, Saratoga Springs (NY), 1992. Springer–Verlag.

[Karnaugh, 1953] G. Karnaugh. The map method for synthesis of combinational logic circuits. *AIEE Transactions on Communications and Electronics*, 72:593–599, 1953.

[Kautz and Selman, 1991] H.A. Kautz and B. Selman. Hard problems for simple default logic. *Artificial Intelligence*, 49:243–279, 1991.

[Kean and Tsiknis, 1988] A. Kean and G. Tsiknis. Clause management systems. technical report 88-21, The University of British Columbia – Department of Computer Science, 1988.

[Kean and Tsiknis, 1990a] A. Kean and G. Tsiknis. Assumption based reasoning and clause management systems. technical report 90-9, The University of British Columbia – Department of Computer Science, 1990.

[Kean and Tsiknis, 1990b] A. Kean and G. Tsiknis. An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, 9:185–206, 1990.

[Kean and Tsiknis, 1992] A. Kean and G. Tsiknis. Assumption-based reasoning and clause management systems. *Computational Intelligence*, 8(1):1–24, 1992.

[Kean and Tsiknis, 1993] A. Kean and G. Tsiknis. Clause management systems (CMS). *Computational Intelligence*, 9(1):11–40, 1993.

[Kean, 1992] A. Kean. *A formal characterization of a domain independent abductive reasoning system.* PhD thesis, University of British Columbia, 1992.

[Kohlas *et al.*, 1999a] J. Kohlas, R. Haenni, and N. Lehmann. Assumption-based reasoning and probabilistic argumentation systems. In D. Gabbay and Ph. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5, chapter 2: Algorithms for Defeasible and Uncertain Reasoning. Kluwer Academic Press, 1999.

[Kohlas *et al.*, 1999b] J. Kohlas, S. Moral, and R. Haenni. Propositional information systems. *Journal of Logic and Computation*, 1999. (to appear).

[Kolaitis and Papadimitriou, 1988] P.G. Kolaitis and Ch.H. Papadimitriou. Some computational aspects of circumscription. In *Proc. of the* 7^{th} *National Conference on Artificial Intelligence (AAAI'88)*, pages 465–469, St Paul (MN), 1988.

[Kolaitis and Papadimitriou, 1990] P.G. Kolaitis and Ch.H. Papadimitriou. Some computational aspects of circumscription. *Journal of the Association for Computing Machinery*, 37:1–14, 1990.

[Konolige, 1992] K. Konolige. Abduction versus closure in causal theories. *Artificial Intelligence*, 53:255–272, 1992.

[Kowalski and Kuhner, 1971] R. Kowalski and D. Kuhner. Linear resolution with selection function. *Artificial Intelligence*, 2:227–260, 1971.

[Krishnaprasad, 1988] T. Krishnaprasad. On the computability of circumscription. *Information Processing Letters*, 27:237–243, 1988.

[Kuznetsov, 1983] V. Kuznetsov. On the lower estimate of the length of the shortest disjunctive normal forms for almost all boolean functions. *Veroiatnostnye Metody v. Kibernetiki*, 19:44–47, 1983. (in russian).

[Lang and Marquis, 1998] J. Lang and P. Marquis. Complexity results for independence and definability. In *Proc. of the* 6^{th} *International Conference on Knowledge Representation and Reasoning (KR'98)*, Trento, 1998. 356–367.

[Lang *et al.*, 1999] J. Lang, P. Liberatore, and P. Marquis. The complexity of propositional independence. Technical report, CRIL/Université d'Artois, 1999. in preparation.

[Lee *et al.*, 1985] W.S. Lee, D.L. Grosh, F.A. Tillman, and C.H. Lie. Fault tree analysis, methods and applications: a review. *IEEE Transactions on Reliability*, 34:194–303, 1985.

[Lee, 1967] R.C.T. Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms.* PhD thesis, University of California, Berkeley (CA), 1967.

[Lifschitz, 1985] V. Lifschitz. Computing circumscription. In *Proc. of the* 9^{th} *International Joint Conference on Artificial Intelligence (IJCAI'85)*, pages 121–127, Los Angeles (CA), 1985.

[Lin and Reiter, 1994] F. Lin and R. Reiter. Forget it! In *Proc. of the AAAI Fall Symposium on Relevance*, pages 154–159, New Orleans, 1994.

[Loveland, 1978] D.W. Loveland. *Automated theorem proving: a logical basis.* Noth-Holland, Amsterdam, 1978.

[Madre and Coudert, 1991] J.-C. Madre and O. Coudert. A logically complete reasoning maintenance system based on a logical constraint solver. In *Proc. of the* 12^{th} *International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 294–299, Sydney, 1991. Extended version as BULL research report "A complete reasoning maintenance system based on typed decision graphs", Louveciennes, 1–26.

[Madre and Coudert, 1994] J.-C. Madre and O. Coudert. Une approche intentionnelle du calcul des implicants premiers et essentiels des fonctions booléennes. *Informatique théorique et Applications*, 28(2), 1994.

[Marquis and Sadaoui, 1996] P. Marquis and S. Sadaoui. A new algorithm for computing theory prime implicates compilations. In *Proc. of the* 13^{th} *National Conference on Artificial Intelligence (AAAI'96)*, pages 504–509, Portland (OR), 1996.

[Marquis, 1991] P. Marquis. *Contribution à l'étude des méthodes de construction d'hypothèses en intelligence artificielle.* Thèse d'université, Université de Nancy 1, 1991. (in french).

[Marquis, 1993] P. Marquis. Skeptical abduction. *International Journal on Artificial Intelligence Tools*, 2(4):511–540, 1993.

[Marquis, 1995] P. Marquis. Knowledge compilation using theory prime implicates. In *Proc. of the* 14^{th} *International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 837–843, Montreal, 1995.

[McCarthy, 1980] J. McCarthy. Circumscription: a form of non–monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[McCarthy, 1986] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[McCluskey, 1956] E.J. McCluskey. Minimization of boolean functions. *Bell System Technical Journal*, 35(6):1445–1446, 1956.

[McMullen and Shearer, 1986] C. McMullen and J. Shearer. Prime implicants, minimum covers and the complexity of logic simplification. *IEEE Transactions on Computers*, 35(8):761–762, 1986.

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

[Minker, 1982] J. Minker. On indefinite databases and the closed world assumption. In *Proc. of the* 6^{th} *Conference on Automated Deduction (CADE'82)*, pages 292–308, 1982.

[Murray and Rosenthal, 1993] N.V. Murray and E. Rosenthal. Dissolution: making paths vanish. *Journal of the Association for Computing Machinery*, 40(3):504–535, 1993.

[Nebel, 1992] B. Nebel. *Belief Revision*, chapter Syntax-based approaches to belief revision, pages 52–88. Number 29 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

[Necula, 1967] N.N Necula. A numerical procedure for determination of the prime implicants of a boolean function. *IEEE Transactions on Electronic Computers*, EC–16:687–689, 1967.

[Nelson, 1955] R.J. Nelson. Simplest normal truth functions. *Journal of Symbolic Logic*, 20:105–108, 1955.

[Ngair, 1993] T.H. Ngair. A new algorithm for incremental prime implicate generation. In *Proc. of the* 13^{th} *International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 46–51, Chambéry, 1993.

[Palopoli *et al.*, 1999] L. Palopoli, F. Pirri, and C. Pizzuti. Algorithms for the selective enumeration of prime implicants. *Artificial Intelligence*, 111:41–72, 1999.

[Papadimitriou, 1994] Ch. H. Papadimitriou. *Computational complexity.* Addison–Wesley, 1994.

[Pizzuti, 1996] C. Pizzuti. Computing prime implicants by integer programming. In *Proc. of the* 8^{th} *IEEE International Conference on Tools with Artificial Intelligence (ICTAI'96)*, pages 332–336, Toulouse, 1996.

[Poole, 1988] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[Poole, 1989] D. Poole. Explanation and prediction: an architecture for default and abductive reasoning. *Computational Intelligence*, 5:97–110, 1989.

[Provan, 1990] G.M. Provan. The computational complexity of multiple-context truth maintenance systems. In *Proc. of the* 9^{th} *European Conference on Artificial Intelligence (ECAI'90)*, pages 522–527, Stockholm, 1990.

[Przymusinski, 1989] T.C. Przymusinski. An algorithm to compute circumscription. *Artificial Intelligence*, 38:49–73, 1989.

[Quine, 1952] W.V.O. Quine. The problem of simplifying truth functions. *American Mathematical Monthly*, 59:521–531, 1952.

[Quine, 1955] W.V.O. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62:627–631, 1955.

[Quine, 1959] W.V.O. Quine. On cores and prime implicants of truth functions. *American Mathematical Monthly*, 66:755–760, 1959.

[Rabinov, 1989] A. Rabinov. A generalization of collapsible cases of circumscription. *Artificial Intelligence*, 38:111–117, 1989.

[Raiman and de Kleer, 1992] O. Raiman and J. de Kleer. A minimality maintenance system. In *Proc. of the* 3^{rd} *International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 532–538, Cambridge (MA), 1992.

[Ramesh and Murray, 1993] A. Ramesh and N.V. Murray. Non-clausal deductive techniques for computing prime implicants and prime implicates. In *Proc. of the* 4^{th} *International Conference on Logic Programming and Automated Reasoning (LPAR'93)*, pages 277–288, St Petersburg, 1993.

[Ramesh *et al.*, 1997] A. Ramesh, G. Becker, and N.V. Murray. CNF and DNF considered harmful for computing prime implicants/implicates. *Journal of Automated Reasoning*, 18:337–356, 1997.

[Reiter and de Kleer, 1987] R. Reiter and J. de Kleer. Foundations of assumption-based truth maintenance systems: preliminary report. In *Proc. of the* 6^{th} *National Conference on Artificial Intelligence (AAAI'87)*, pages 183–188, Seattle (WA), 1987.

[Reiter, 1978] R. Reiter. *On closed world data bases*, pages 119–140. Plenum Press, 1978.

[Reiter, 1980] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[Robinson, 1965] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

[Rymon, 1992] R. Rymon. Search through systematic search enumeration. In *Proc. of the* 3^{rd} *International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 539–550, Cambridge (MA), 1992.

[Rymon, 1994] R. Rymon. An SE-tree-based prime implicant generation algorithm. *Annals of Mathematics and Artificial Intelligence*, 11, 1994. special issue on model-based diagnosis.

[Selman and Levesque, 1990] B. Selman and H. Levesque. Abductive and default reasoning: a computational core. In *Proc. of the* 8^{th} *National Conference on Artificial Intelligence (AAAI'90)*, pages 343–348, Boston (MA), 1990.

[Shannon, 1949] C.E. Shannon. The synthesis of two–terminal switching circuits. *Bell System Technical Journal*, 28(1):59–98, 1949.

[Shiny and Pujari, 1998] A.K. Shiny and A.K. Pujari. Computation of prime implicants using matrix and paths. *Journal of Logic and Computation*, 8:135–145, 1998.

[Shiny and Pujari, 1999] A.K. Shiny and A.K. Pujari. An efficient algorithm to generate prime implicants. *Journal of Automated Reasoning*, 22:149–170, 1999.

[Shoham, 1987] Y. Shoham. A semantical approach to nonmonotonic logics. In *Proc. of the* 2^{nd} *IEEE Symposium on Logic in Computer Science (LICS'87)*, pages 275–279, Washington (DC), 1987.

[Shostak, 1976] R. Shostak. Refutation graphs. *Artificial Intelligence*, 7:51–64, 1976.

[Siegel and Forget, 1996] P. Siegel and L. Forget. A representation theorem for preferential logics. In *Proc. of the* 5^{th} *International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, Cambridge (MA), 1996.

[Siegel, 1987] P. Siegel. *Représentation et utilisation des connaissances en calcul propositionnel.* Thèse d'etat, Université d'Aix–Marseille 2, 1987. (in french).

[Slagle *et al.*, 1969] J.R. Slagle, C.L. Chang, and R.C.T. Lee. Completeness theorems for semantics resolution in consequence finding. In *Proc. of the* 1^{st} *International Joint Conference on Artificial Intelligence (IJCAI'69)*, pages 281–285, Washington (DC), 1969.

[Slagle *et al.*, 1970] J.R. Slagle, C.L. Chang, and R.C.T. Lee. A new algorithm for generating prime implicants. *IEEE Transactions on Computers*, C-19:304–310, 1970.

[Slagle, 1967] J.R. Slagle. Automatic theorem proving with renamable and semantic resolution. *Journal of the Association for Computing Machinery*, 12:687–697, 1967.

[Socher, 1991] R. Socher. Optimizing the clausal normal form transformation. *Journal of Automated Reasoning*, 7:325–336, 1991.

[Stillman, 1992] J. Stillman. The complexity of propositional default logics. In *Proc. of the* 10^{th} *National Conference on Artificial Intelligence (AAAI'92)*, pages 794–799, San Jose (CA), 1992.

[Strzemecki, 1992] T. Strzemecki. Polynomial-time algorithm for generation of prime implicants. *Journal of Complexity*, 8:37–63, 1992.

[Tison, 1967] P. Tison. Generalization of consensus theory and application to the minimization of boolean functions. *IEEE Transactions on Electronic Computers*, EC–16:446–456, 1967.

[Tsiknis and Kean, 1988] G. Tsiknis and A. Kean. Clause management systems (CMS). technical report 88-21, The University of British Columbia – Department of Computer Science, 1988.

[Uribe and Stickel, 1994] T.E. Uribe and M.E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In *Proc. of the* 1^{st} *International Conference on Constraints in Computational Logics (CCL'94)*, 1994.

[Wos *et al.*, 1967] L. Wos, G.A. Robinson, and D.F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the Association for Computing Machinery*, 12:536–541, 1967.

[Yahya and Henschen, 1985] A. Yahya and L.J. Henschen. Deduction in non-Horn databases. *Journal of Automated Reasoning*, 1:141–160, 1985.

Appendix

Proposition 3.1
See e.g., Proposition 1 in [Marquis, 1993]. ■

Proposition 3.2
Easy consequence of, e.g., Proposition 3 from [Marquis, 1993]. ■

Proposition 3.3
Assume that there exists $\pi \in PI(\Sigma)$ s.t. π contains a literal l Σ is independent from which. $\Sigma \models \pi$ holds iff $\models (\Sigma \Rightarrow \pi)$ holds. If $\models (\Sigma \Rightarrow \pi)$ holds, then $\models (\Sigma_{l \leftarrow false} \Rightarrow \pi_{l \leftarrow false})$ holds. In this case, $\Sigma_{l \leftarrow false} \models \pi_{l \leftarrow false}$ holds as well. When Σ is independent from l, we have $\Sigma \models \Sigma_{l \leftarrow false}$. Subsequently, $\Sigma \models \pi_{l \leftarrow false}$ holds as well. This contradicts the fact that π is a prime implicate of Σ since $\pi_{l \leftarrow false} \models \pi$ holds, unless $\pi_{l \leftarrow false} \equiv \pi$ holds as well. Hence, π is equivalent to a clause that does not contain l. ■

Proposition 3.4
See e.g., Proposition 3 from [Marquis, 1993]. ■

Proposition 3.5
See [Quine, 1955]. ■

Proposition 3.6
See e.g., Corollary to Proposition 7 in [Marquis, 1993]. ■

Proposition 3.7
It is sufficient to prove that every resolvent from clauses of $min(\{\Sigma\} \cup \{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\} \cup \{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}, \models)$ is a logical consequence of a clause in that set. Then, the correctness of Tison's algorithm for generating prime implicates concludes the proof.

Let π_1 and π_2 be two clauses from $min(\{\Sigma\} \cup \{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\} \cup \{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}, \models)$ that can be resolved upon. Clearly enough, π_1 and π_2 neither contain Σ nor $\neg\Sigma$. For each π_i $(i \in 1 \,..\, 2)$, there are two possibilities:

- π_i is the resolvent of a clause π'_i from $PI(\Sigma)$ and Σ. π'_i is s.t. $\pi'_i \equiv (\pi \vee \neg\Sigma)$ (situation A).
- π_i is a prime implicate of Σ (situation B).

Since π_1 and π_2 play symmetric roles, three cases are to be considered, only:

- π_1 and π_2 are both of type B. Then their resolvent π_3 is implied by a prime implicate π of Σ that contains neither Σ nor $\neg\Sigma$. Hence, π

belongs to $min(\{\Sigma\} \cup \{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\} \cup \{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}, \models)$. Subsequently, π_3 will be removed in the minimization step, hence generating it is useless.

- π_1 is of type A and π_2 is of type B. If π_1 and π_2 can be resolved upon, this is also the case for the prime implicate of Φ $\pi'_1(\equiv (\pi_1 \vee \neg\Sigma))$ and π_2 and Σ is not the literal resolved upon. The resolvent π'_3 of π'_1 and π_2 can be written as $(\pi_1 \setminus \{l\}) \cup \{\neg\Sigma\} \cup (\pi_2 \setminus \{\neg l\})$ when the resolvent π_3 of π_1 and π_2 can be written as $(\pi_1 \setminus \{l\}) \cup (\pi_2 \setminus \{\neg l\})$ (here, clauses are considered as sets of literals for simplicity). Since both π'_1 and π_2 are implicates of Φ, there exists $\pi \in PI(\Sigma)$ s.t. $\pi \models \pi'_3$ holds. If π contains $\neg\Sigma$, then there exists a clause π' that is at least as logically strong as $\pi \setminus \{\neg\Sigma\}$ in $min(\{\Sigma\} \cup \{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\} \cup \{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}, \models)$. Subsequenlty, π' implies π_3, so generating π_3 is useless. Otherwise, π does not contain $\neg\Sigma$. In this case, π is logically stronger than π_3. Since neither π_1 nor π_2 contains Σ, this is also the case for π. Hence, in $min(\{\Sigma\} \cup \{\pi; (\neg\Sigma \vee \pi) \in PI(\Phi)$ and $\neg\Sigma \not\models \pi\} \cup \{\pi; \pi \in PI(\Phi)$ and $\neg\Sigma \not\models \pi$ and $\Sigma \not\models \pi\}, \models)$, we can find a clause π' that is at least as strong than π (it is strictly stronger than π if there exists a clause $(\pi' \vee \neg\Sigma)$ in $PI(\Phi)$ s.t. $\pi' \models \pi$. Since $\pi' \models \pi_3$ holds, generating π_3 is useless.

- π_1 and π_2 are both of type A. Then there exists π'_1 (resp. π'_2) in $PI(\Phi)$ s.t. $\pi'_1 \equiv (\pi_1 \vee \neg\Sigma)$ and $\pi'_2 \equiv (\pi_2 \vee \neg\Sigma)$. The resolvent π'_3 of π'_1 and π'_2 can be written as $(\pi_1 \setminus \{l\}) \cup \{\neg\Sigma\} \cup (\pi_2 \setminus \{\neg l\})$ when the resolvent π_3 of π_1 and π_2 can be written as $(\pi_1 \setminus \{l\}) \cup (\pi_2 \setminus \{\neg l\})$. The previous case is recovered, *mutatis mutandis*.

$\blacksquare$

Proposition 3.8
See Proposition 2.9 from [Inoue, 1992]. $\blacksquare$

Proposition 3.9
See e.g., Proposition 4 from [Marquis, 1993]. $\blacksquare$

Proposition 3.10
See [Madre and Coudert, 1991]. $\blacksquare$

Proposition 3.11
Easy consequence of the definition of L-prime implicates. $\blacksquare$

Proposition 3.12
By induction on $|L|$. The base case where $|L| = 0$ is trivial. Let us now assume that the proposition holds for every $|L| \leq k$ and show that it re-

mains true when $|L| = k+1$. Let $L = L' \cup \{l\}$. By induction hypothesis, we can assume that $forget(\Sigma, L')$ is the most general consequence of Σ that is independent from L'. For the sake of simplicity, let us note Σ' this formula. It remains to show that $forget(\Sigma, L) = forget(forget(\Sigma, L'), \{l\}) = forget(\Sigma', \{l\})$ is the most general consequence of Σ' that is independent from l. Two cases are to be considered:

- $l = x$. $forget(\Sigma', x)$ is independent from x iff $forget(\Sigma', x) \models forget(\Sigma', x)_{x \leftarrow false}$ holds. We have $(\Sigma'_{x \leftarrow true} \vee (\neg x \wedge \Sigma'_{x \leftarrow false}))_{x \leftarrow false} \equiv (\Sigma'_{x \leftarrow true} \vee \Sigma'_{\leftarrow false})$. This formula clearly is a logical consequence of $forget(\Sigma', \{x\})$. Hence $forget(\Sigma', x)$ is independent from x. $forget(\Sigma', x)$ is a logical consequence of Σ'. Indeed, for every $\Sigma' \in PROP_{PS}$, $x \in PS$, we have $\Sigma' \equiv (x \wedge \Sigma'_{x \leftarrow true}) \vee (\neg x \wedge \Sigma'_{x \leftarrow false})$. It remains to show that every logical consequence Φ of Σ' that is independent from x is a logical consequence of $forget(\Sigma', x)$. Let Φ any formula s.t. $\Sigma' \models \Phi$ holds and $\Phi \models \Phi_{x \leftarrow false}$. Since $\Sigma' \equiv (x \wedge \Sigma'_{x \leftarrow true}) \vee (\neg x \wedge \Sigma'_{x \leftarrow false})$ holds, we have $\Sigma' \models \Phi$ iff both $(x \wedge \Sigma'_{x \leftarrow true}) \models \Phi$ and $(\neg x \wedge \Sigma'_{x \leftarrow false}) \models \Phi$ hold. Thus, in order to show that $forget(\Sigma', x) \models \Phi$ holds, it is sufficient to prove that $\Sigma'_{x \leftarrow true} \models \Phi$ holds. Suppose that is not the case. Then there exists an implicant γ of $\Sigma'_{x \leftarrow true}$ that is not an implicant of Φ. However, since $(x \wedge \Sigma'_{x \leftarrow true}) \models \Phi$ holds, we know that $(x \wedge \gamma)$ is an implicant of Φ. Since Φ is equivalent to the conjunction of its prime implicates, there necessarily exists a prime implicate π of Φ s.t. $(x \wedge \gamma) \models \pi$ and $\gamma \not\models \pi$ hold. This imposes that x belongs to π but no literal of γ belongs to π. By construction, $\pi_{x \leftarrow false}$ is an implicate of $\Phi_{x \leftarrow false}$ and $\pi_{x \leftarrow false}$ is strictly stronger than π. Since $\Phi \models \Phi_{x \leftarrow false}$ holds, $\Phi \models \pi_{x \leftarrow false}$ holds as well. This contradicts the fact that π is a prime implicate of Φ.

- $l = \neg x$. The demonstration is similar, *mutatis mutandis*.

■

Proposition 3.13

- $\subseteq$. Let π be any L-prime implicate of Σ. Then for every $l \in Lit(\Sigma) \setminus L$, l does not occur in π. Hence, π is independent from every such l. Accordingly, π is a logical consequence of $forget(\Sigma, Lit(\Sigma) \setminus L)$. Subsequently, there exists a prime implicate π' of $forget(\Sigma, Lit(\Sigma) \setminus L)$ s.t. $\pi' \models \pi$ holds. Since $\Sigma \models forget(\Sigma, Lit(\Sigma) \setminus L)$ holds, there exists $\pi'' \in PI(\Sigma)$ s.t. $\pi'' \models \pi'$ holds. Hence, $\pi'' \models \pi$ holds. Since every L-prime implicate of Σ is a prime implicate of Σ, we necessarily have $\pi'' \equiv \pi$. Hence, $\pi \equiv \pi'$ holds as well.

- $\supseteq$. Let π be a prime implicate of $forget(\Sigma, Lit(\Sigma) \setminus L)$. π is a L-implicate of Σ since π contains only literals from L and $\Sigma \models forget(\Sigma, Lit(\Sigma) \setminus$

L) holds. Therefore, there exists $\pi' \in PI_L(\Sigma)$ s.t. $\pi' \models \pi$ holds. Since π' is independent from $Lit(\Sigma) \setminus L$ (by construction), $forget(\Sigma, Lit(\Sigma) \setminus L) \models \pi'$ holds. Hence there exists $\pi'' \in PI(forget(\Sigma, Lit(\Sigma) \setminus L)$ s.t. $\pi'' \models \pi'$ holds. As a consequence, $\pi'' \equiv \pi$ holds and $\pi \equiv \pi'$ holds as well.

■

Proposition 3.14
Easy consequences of the definition of $(\exists x)\Sigma$. ■

Proposition 3.15
See Theorem 5 in [del Val, 1999]. ■

Proposition 3.16
Easy consequence of Proposition 3.9 and Proposition 3.11. ■

Proposition 3.17
Easy consequence of Proposition 3.8 and Proposition 3.11. ■

Proposition 3.18
Let π be any implicate of Φ that also is a prime implicate of $\Phi \wedge \Sigma$. Proposition 3.4 shows that there must exist a prime implicate π_Φ of Φ s.t. $\pi_\Phi \models \pi$ holds. Since $\Phi \wedge \Sigma \models \Phi$ and $\Phi \models \pi_\Phi$ hold, it is the case that $\Phi \wedge \Sigma \models \pi_\Phi$ holds. Proposition 3.4 imposes that there exists a prime implicate $\pi_{\Phi \wedge \Sigma}$ s.t. $\pi_{\Phi \wedge \Sigma} \models \pi_\Phi$ holds. Since $\pi_\Phi \models \pi$ holds, we also have $\pi_{\Phi \wedge \Sigma} \models \pi$. Since both $\pi_{\Phi \wedge \Sigma}$ and π are prime implicates of $\Phi \wedge \Sigma$, we must have $\pi_{\Phi \wedge \Sigma} \equiv \pi$. This implies that $\pi_\Phi \equiv \pi$ holds, i.e., π is a prime implicate of Φ. ■

Proposition 3.19
See Proposition 2.8 from [Inoue, 1992]. ■

Proposition 3.20
See Corollary 4 from [Reiter and de Kleer, 1987]. ■

Proposition 3.21
See Theorem 5 from [Reiter and de Kleer, 1987] (see also Theorem 4.1 from [Kean and Tsiknis, 1988], [Kean and Tsiknis, 1993]). ■

Proposition 3.22
Proposition 3.18 shows that $PI^\Phi(\Sigma_1 \vee ... \vee \Sigma_k) = \{\pi \in PI((\Sigma_1 \vee ... \vee \Sigma_k) \wedge \Phi); (\forall \pi_\Phi \in PI(\Phi))(\pi \not\equiv \pi_\Phi)\}$. But Proposition 3.9 shows that $PI((\Sigma_1 \vee ... \vee \Sigma_k) \wedge \Phi) = min(\{(\pi_{\Sigma_1} \vee ... \vee \pi_{\Sigma_k}); \pi_{\Sigma_1} \in PI(\Sigma_1 \wedge \Phi), ..., \pi_{\Sigma_k} \in PI^(\Sigma_k \wedge \Phi)\}, \models)$. Let π_Φ be a prime implicate of Φ. Every clause that is a logical conse-

quence of a clause of the form $\pi_\Phi \vee ...$ is a Φ–implicate. Hence, it will be removed once distribution is achieved. Instead of removing such clauses afterwards, it is possible to avoid generating them, i.e., to remove the prime implicates of Φ from every $PI(\Sigma_i \wedge \Phi)$ in the previous equation. See Theorem 6.2 from [Kean and Tsiknis, 1988], [Kean and Tsiknis, 1993] for more details. ■

Proposition 3.23
Easy consequence of Proposition 3.18 and Proposition 3.11. ■

Proposition 3.24
See Proposition 2.8 from [Inoue, 1992]. ■

Proposition 3.25
See Theorem 4.7 from [Inoue, 1992]. ■

Proposition 3.26
Easy consequence of Proposition 3.22 and the fact that the $\langle L, \Phi\rangle$-prime implicates of Σ are its Φ-prime implicates built up solely from L. ■

Proposition 3.27
In order to check whether π is a prime implicate of Σ, it is sufficient to first check whether $\Sigma \models \pi$ holds (formally, this is an instance of $UNSAT$, the complement of SAT). If no, π is not a prime implicate of Σ. Otherwise, we reduce π by removing from it every occurrence of the $false$ symbol, and every redundant occurrence of every literal (this reduction can be easily achieved in time polynomial in the size of π). Let l be any literal from the resulting π and π_l be the clause obtained from π by removing the occurrence of l. π is a prime implicate of Σ iff for every l, we have $\Sigma \not\models \pi_l$ (formally, every test is an instance of SAT). Clearly enough, only a linear number of calls to an NP–oracle are required to check this property. Since several SAT instances can be condensed into a single SAT instance (using some renamings so as to make the instances independent one another), checking whether an implicate π of Σ is prime can be encoded in polynomial time as a single instance of SAT. Hence, PRIME IMPLICATE can be polynomially reduced to $SAT - UNSAT$, which is the canonical BH_2-complete problem. This shows the membership of PRIME IMPLICATE to BH_2. Conversely, let us consider the mapping M that associates to every ordered pair $\langle\phi, \psi\rangle$ of propositional formulas the following instance of PRIME IMPLICATE: $\langle(\phi \wedge new) \vee \psi, new\rangle$, where new is a variable from $PS \setminus Var(\phi \wedge \psi)$. Clearly enough, $M(\langle\phi, \psi\rangle)$ can be computed in time polynomial in the size of the input. Moreover, we have $\langle\phi, \psi\rangle \in SAT - UNSAT$ i.e., ϕ is satisfiable and ψ is not iff $M(\langle\phi, \psi\rangle) \in$ PRIME IMPLICATE. This shows the completeness of PRIME IMPLICATE for BH_2. ■

Proposition 3.28
Since the notion of L–prime implicate can be viewed as a generalization of the basic notion of prime implicate, the decision problem L–PRIME IMPLICATE is BH_2–hard. It also belongs to BH_2; indeed, in order to check whether or not π is a L–prime implicate of Σ, we check whether π is a prime implicate of Σ (as shown before, this is in BH_2) and if so, we check whether π is equivalent to a clause containing literals from L, only (which can be done easily in time polynomial in $|\Sigma| + |L|$). If both tests succeed, then π is a L–prime implicate of Σ, otherwise, it is not. ■

Proposition 3.29
If $PI_L(\Sigma)$ is not empty, we can easily guess a clause π and check in time polynomial in the size of the input that π is a L–prime implicate of Σ (since L–PRIME IMPLICATE is BH_2–complete, this can easily be achieved using a polynomial number of calls to an NP–oracle). This shows the membership of the complementary problem to Σ_2^p. Π_2^p–hardness comes from the following polynomial reduction from the validity problem of a formula from $QBF_{2,\forall}$ to the vacuity of a set of L–prime implicates: let M the mapping that associates to every quantified boolean formula of the form $\forall A \exists B \Sigma$ (where $\Sigma \in PROP_{PS}$ and $\{A, B\}$ is a partition of $Var(\Sigma)$) the following ordered pair $\langle \Sigma \wedge new, L_A \rangle$ where $new \in PS \setminus Var(\Sigma)$. $M(\forall A \exists B \Sigma)$ can be computed in time polynomial in the size of the input. $\forall A \exists B \Sigma$ is valid iff every prime implicate of $\Sigma \wedge new$ contains a literal from $L_B \cup \{new\}$. In other words, $\forall A \exists B \Sigma$ is valid iff $PI_{L_A}(\Sigma \wedge new)$ is empty. Since checking the validity of a formula from $QBF_{2,\forall}$ is the canonical Π_2^p–complete problem, the Π_2^p–hardness of checking whether or not a L–prime implicate of a formula exists follows. Intractability is also the case for the problem of checking whether or not there is no non-valid L–implicate of Σ since Σ has no L–prime implicate iff Σ has no L–implicate, except tautologies. ■

Proposition 3.30
The membership to Σ_2^p is easy since verifying whether a guess π is L–prime implicate of Σ equivalent to a clause containing l is in BH_2. Hardness can be proved by the following transformation. Let M be a mapping that associates to every instance $\langle \Sigma, L \rangle$ of L–PRIME IMPLICATE VACUITY the following triple $\langle \Sigma \vee new, L \cup \{new\}, new \rangle$, where $new \in PS \setminus (Var(\Sigma) \cup L)$. $M(\langle \Sigma, L \rangle)$ can be easily computed in polynomial time in the size of the input. Moreover, $PI_L(\Sigma) \neq \emptyset$ iff $M(\langle \Sigma, L \rangle)$ is a "yes"-instance of L–PRIME IMPLICATE RELEVANCE. Since the complement of L–PRIME IMPLICATE VACUITY is Σ_2^p–complete, the Σ_2^p–hardness of L–PRIME IMPLICATE RELEVANCE follows. ■

Proposition 3.31
In order to show the membership of the complementary problem to Σ_2^p, it

is sufficient to show that checking whether a guessed clause π is a L–prime implicate of Σ and is not equivalent to a clause containing l can be achieved using a polynomial number of calls to an NP–oracle. This is trivial since we have shown that L–PRIME IMPLICATE is BH_2–complete. Π_2^p–hardness is a consequence of the following polynomial reduction: let M be a mapping that associates to every instance $\langle \Sigma, L \rangle$ of L–PRIME IMPLICATE VACUITY the following triple $\langle \Sigma \wedge new, L \cup \{new\}, new \rangle$, where $new \in PS \setminus Var(\Sigma)$. It is easy to see that $M(\langle \Sigma, L \rangle)$ can be computed in time polynomial in the input size. Moreover, it is easy to show that every L–prime implicate of Σ is equivalent to a $(L \cup \{new\})$–prime implicate of $\Sigma \wedge new$, unless Σ is valid. We can legitimately assume that Σ is not valid, since the proof of Proposition 3.29 shows that checking that $PI_L(\Sigma) = \emptyset$ remains Π_2^p-complete in the case where Σ is not valid. If $PI_L(\Sigma) = \emptyset$, then $PI_{L \cup \{new\}}(\Sigma \wedge new) = \{new\}$. If $PI_L(\Sigma) \neq \emptyset$, then there exists a prime implicate γ of Σ s.t. $Lit(\gamma) \subseteq L$. We have $\gamma \in PI_{L \cup \{new\}}(\Sigma \wedge new)$ since we have assumed that Σ is not valid. By construction, we have $new \notin Lit(\gamma)$. Hence, we have $PI_L(\Sigma) = \emptyset$ iff $M(\langle \Sigma, L \rangle)$ is a "yes"-instance of L–PRIME IMPLICATE NECESSITY, which shows the Π_2^p hardness of L–PRIME IMPLICATE NECESSITY. ■

Proposition 3.32

The algorithm for checking whether a given clause is a prime implicate of Σ can be easily suited to the Φ–prime implicate situation. Indeed, it is sufficient to first check whether π is an implicate of $\Phi \wedge \Sigma$ (formally, this is an instance of $UNSAT$). Then, we have to check whether $\Phi \not\models \pi$ (formally, this is an instance of SAT). Assuming that π has been simplified by removing occurrences of $false$ and redundant occurrences of literals (which can be done in polynomial time), it remains to check that each clause obtained by removing a literal from the simplified π is not an implicate of $\Phi \wedge \Sigma$ (note that if π is not an implicate of Φ then no clause logically stronger than π can be an implicate of Φ). All these tests can be gathered into a single SAT instance, renaming variables in a systematic way (and the size of this instance is polynomially bounded by the size of the input). This shows the membership of Φ-PRIME IMPLICATE to BH_2. Hardness comes from BH_2-hardness of PRIME IMPLICATE. ■

Proposition 3.33

Easy consequence from the fact that $PI^{\Phi}(\Sigma) = \emptyset$ iff $\Phi \wedge \neg \Sigma$ is unsatisfiable. ■

Proposition 3.34

Membership can be proved easily from Proposition 3.32 (checking that π is built up from literals of L, only, is computationally easy and does not add an additional source of complexity). Hardness comes from BH_2-hardness of Φ-PRIME IMPLICATE. ■

Proposition 3.35
Membership can be proved easily from Proposition 3.29 (checking whether $\Phi \not\models \pi$ can be easily accomplished in non deterministic polynomial time). Hardness comes from the Π_2^p-hardness of L-PRIME IMPLICATE VACUITY, the restriction to the case where Φ is valid (in this situation, the $\langle L, \Phi\rangle$–prime implicates of Σ are its L–prime implicates except when Σ is valid - in this case, $PI_L^\Phi(\Sigma) = \emptyset$). ■

Proposition 3.36
Membership can be proved easily from Proposition 3.30 (checking whether $\Phi \not\models \pi$ can be easily accomplished in non deterministic polynomial time). Hardness comes from the Π_2^p-hardness of L-PRIME IMPLICATE RELEVANCE, the restriction to the case where Φ is valid (in this situation, the $\langle L, \Phi\rangle$–prime implicates of Σ are its L–prime implicates except when Σ is valid - in this case, $PI_L^\Phi(\Sigma) = \emptyset$, hence the answer is no). ■

Proposition 3.37
Membership can be proved easily from Proposition 3.31 (checking whether $\Phi \not\models \pi$ can be easily accomplished in non deterministic polynomial time). Hardness comes from the Π_2^p-hardness of L-PRIME IMPLICATE NECESSITY, the restriction to the case where Φ is valid (in this situation, the $\langle L, \Phi\rangle$–prime implicates of Σ are its L–prime implicates except when Σ is valid - in this case, $PI_L^\Phi(\Sigma) = \emptyset$, hence the answer is yes). ■

Proposition 4.1
Easy consequence of the substitution theorem of propositional logic. See e.g., Theorem 4.3.1 in [Kean, 1992]. ■

Proposition 4.2
See e.g., Lemma 4.1 from [Kean and Tsiknis, 1990a], [Kean and Tsiknis, 1992] (see also Proposition 3.2 from [Inoue, 1992]). ■

Proposition 4.3

- Let us first prove that $Interpretation(T) = \{\bar{\gamma} \subseteq H | \gamma \in MHS(Nogood(T))\}$. First note that for every subset γ of H, we have $\Phi \wedge \gamma$ is consistent iff $\forall N \in Nogood(T)$, $N \not\subseteq \gamma$ iff $\forall N \in Nogood(T)$, $N \cap \bar{\gamma} \neq \emptyset$ iff $\bar{\gamma}$ is a hitting set of $Nogood(T)$. Now, $\gamma \in Interpretation(T)$ iff $\Phi \wedge \gamma$ is consistent and $\forall h \in \bar{\gamma}, \Phi \wedge (\gamma \cup \{h\})$ is inconsistent iff $\bar{\gamma}$ is a hitting set of $Nogood(T)$ and $\forall h \in \bar{\gamma}, \overline{(\gamma \cup \{h\})}$ is not a hitting set of $Nogood(T)$ iff $\bar{\gamma}$ is a hitting set of $Nogood(T)$ and $\forall h \in \bar{\gamma}, \bar{\gamma} \setminus \{h\}$ is not a hitting set of $Nogood(T)$ iff $\bar{\gamma}$ is a minimal hitting set of $Nogood(T)$.
- Let us now prove that $Nogood(T) = MHS(\{\bar{\gamma} \subseteq H | \gamma \in Interpretation(T)\}$.

First note that for every subset γ of H, we have $\Phi \wedge \gamma$ is inconsistent iff $\forall I \in Interpretation(T)$, $\gamma \not\subseteq I$ iff $\forall I \in Interpretation(T)$, $\gamma \cap \bar{I} \neq \emptyset$ iff γ is a hitting set of $\{\bar{I} \subseteq H | I \in Interpretation(T)\}$. Now, $\gamma \in Nogood(T)$ iff $\Phi \wedge \gamma$ is inconsistent and $\forall h \in \gamma, \Phi \wedge (\gamma \setminus \{h\})$ is consistent iff γ is a hitting set of $\{\bar{I} \subseteq H | I \in Interpretation(T)\}$ and $\forall h \in \gamma, (\gamma \setminus \{h\})$ is not a hitting set of $\{\bar{I} \subseteq H | I \in Interpretation(T)\}$ iff γ is a minimal hitting set of $\{\bar{I} \subseteq H | I \in Interpretation(T)\}$.

■

Proposition 4.4
See Theorem 4 from [Poole, 1988] (see also Theorem 3.1 from [Kean and Tsiknis, 1990a] [Kean and Tsiknis, 1992]). ■

Proposition 4.5
See e.g., [Cayrol, 1992]. ■

Proposition 4.6
Easy consequence of the substitution theorem of propositional logic. See e.g., Theorem 4.3.1 in [Kean, 1992]. ■

Proposition 4.7
We have $\Phi \wedge \gamma$ inconsistent and $\gamma \subseteq H$ iff $\neg\gamma$ is a $\neg H$-implicate of Φ. $\neg\gamma$ is a minimal $\neg H$-implicate of Φ w.r.t. $\models$ (i.e., a $\neg H$-prime implicate of Φ) iff γ is a term built upon H that is minimal w.r.t. $\models$ and s.t. $\Phi \wedge \gamma$ is inconsistent iff γ is a subset of H that is maximal w.r.t. $\subseteq$ and s.t. $\Phi \wedge \gamma$ is inconsistent. Indeed, since H is a consistent set, $\models$ and $\supseteq$ coincide over (conjunctively-interpreted) subsets of H. See also Lemma 4.3.3 and Corollary 4.3.1 in [Kean, 1992]. ■

Proposition 4.8
See Theorem 7 from [Selman and Levesque, 1990] (see also [Reiter, 1987], [Kean and Tsiknis, 1990a] [Kean and Tsiknis, 1992]). ■

Proposition 4.9
See Theorem 3.2 from [Kean and Tsiknis, 1990a] [Kean and Tsiknis, 1992]. ■

Proposition 4.10
See Theorem 2 from [Minker, 1982] (GCWA), Theorem 14 from [Yahya and Henschen, 1985] (EGCWA), Theorem 3.6 from [Gelfond and Przymusinska, 1986] (CCWA), and Theorem 4.5 from [Gelfond *et al.*, 1989] (ECWA). ■

Proposition 4.11
Easy consequence of Proposition 2.10 (new version) from [Inoue and Helft,

1990] and Theorem 3.2 from [Kean and Tsiknis, 1990a] [Kean and Tsiknis, 1992]. (see also [Etherington, 1987] and [Poole, 1989]). ■

Proposition 4.12
Easy consequences of the definitions of CWA, GCWA, EGCWA, CCWA and ECWA-ffn formulas. ■

Proposition 4.13
Easy consequence of Theorem 2.8 (new version) from [Inoue and Helft, 1990], plus the fact that every formula that is logically stronger than an unexplainable formula is unexplainable as well. ■

Proposition 4.14
Easy consequence of Theorem 3.9 from [Przymusinski, 1989], provided that $PI_{L_P^+ \cup L_Q}(\Phi \wedge \neg\Psi)$ is Φ-equivalent to the conjunction of all MILO-leaves that can be get out from Φ and the negations of literals of Ψ as top clauses. ■

Proposition 4.15
See Theorem 2 from [Raiman and de Kleer, 1992]. ■

Proposition 4.16
Immediate from the definition of minimal support. ■

Proposition 4.17
Immediate from the definition of L-minimal support. ■