ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Artificial Intelligence 172 (2008) 265–299

Artificial Intelligence

www.elsevier.com/locate/artint

# Redundancy in logic II: 2CNF and Horn propositional formulae

Paolo Liberatore

*Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Ariosto 25, 00185 Roma, Italy*

Received 9 August 2006; received in revised form 4 June 2007; accepted 15 June 2007

Available online 27 June 2007

**Abstract**

We report results about the redundancy of formulae in 2CNF form. In particular, we give a slight improvement over the trivial redundancy algorithm and give some complexity results about some problems related to finding Irredundant Equivalent Subsets (I.E.S.) of 2CNF formulae. The problems of checking whether a 2CNF formula has a unique I.E.S. and checking whether a clause in is all its I.E.S.'s are polynomial. Checking whether a 2CNF formula has an I.E.S. of a given size and checking whether a clause is in some I.E.S.'s of a 2CNF formula are polynomial or NP-complete depending on whether the formula is cyclic. Some results about Horn formulae are also reported.

*Keywords:* Propositional logic; Computational complexity; Redundancy

## 1. Introduction

Problems related to redundancy in logic and similar fields has been investigated by a number of authors. Ginsberg [9] and Schmolze and Snyder [25] studied the problem of redundancy of production rules. Several authors investigated the problem of minimizing a formula, with particular emphasis on Horn formulae [1,11,12,19,20,28]. Gottlob and Fermüller [10] gave results about the redundancy of a literal in a first-order clause. Liberatore [17,18] provided complexity results for the CNF case and for some non-classical logics. Büning and Zhao [3] studied the problems of equivalence and extension-equivalence of irredundant formulae. Various authors have studied the problem of minimal unsatisfiability [2,7,21].

In this article, we report about the complexity of some problems related to the redundancy of formulae in 2CNF and Horn form. In particular, the considered problems are that of checking whether a formula is redundant, establishing the minimal size of an Irredundant Equivalent Subset (I.E.S.) of a formula, and checking whether a clause is in some I.E.S.'s of a formula. The first problem is polynomial due to the polynomiality of consistency checking for 2CNF and Horn formulae, but a slight improvement over the trivial algorithm exists. For the two other problems, complexity is shown to be polynomial or NP-complete depending on the structure of the formula.

There are several reasons for checking redundancy of a formula and for finding an irredundant and equivalent subset of it. Most of these reasons apply to all kind of formulae, regardless of whether they are general propositional

*E-mail address:* paolo@liberatore.org.

0004-3702/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2007.06.003

formulae, 2CNF, Horn, or formulae of non-classical logics. Some are peculiar to tractable restrictions such as 2CNF and Horn.

Generally, a logic formula may come either from a user directly encoding a domain of interest or from an automated translation from some formal language. In the first case, the presence of a redundant part of a formula may indicate that some aspects of the domain are either of particular importance or not sufficiently understood. Intentional redundancy of information about a particular part of a domain may indicate a will or need to be sure that part of the domain is correctly encoded. On the other hand, an unintentional redundancy may indicate undetected problems with the encoding, such as multiple people using the same names for expressing different concepts. Simplifying a formula by removing redundancy shows which parts are really needed; this may result in error detection if redundancy was unintentional, and in a confirmation of correctness if it was intentional.

If a formula results from a translation, its redundancy may give the same indications on the information in the language it was originally expressed. However, it may also highlight problems that are introduced by the translation itself. As an example, while each of a set of formulae may be correct by itself, their merging may result in an inconsistent formula because of the different meaning of the variables in different formulae [16]. The same problem may however also generate redundancy. Depending on the particular scenario, redundancy may also results from several sources providing the same information. Either way, detecting redundancy tells something significant.

On the technical side, redundancy has been sometimes intentionally added to formulae to speed-up consistency and entailment solving. Removing redundancy is a way to make such formulae easier to understand to humans. On the other hand, increasing the size of formulae generally makes solving harder. This is particular important for those cases where consistency and entailment are polynomial problems, such as 2CNF and Horn. If a set of clauses originally contains a large number of redundant clauses, removing some redundant ones may provide a speedup due to the reduction in size of the original formula. This gain is useful when the same formula is for example checked for consistency with or entailment to several different formulae, because the cost of redundancy removal is amortized among several consistency or entailment tests.

The results presented in this paper differ from previous work in either the settings or in the specific analyzed problems, or both. Related work regarding different settings are those by Ginsberg [9] and Schmolze and Snyder [25], as they are about production rules and not propositional formulae. Gottlob and Fermüller [10] also worked in a different logic, the first-order one, but also studied a different problem, that or redundancy within a single clause rather than the redundancy of a clause within a set of clauses. Liberatore [18] investigated redundancy for some non-classical logics.

Related work regarding propositional logic can be roughly divided into three classes. First, we have the investigations into making a propositional formula as small as possible while preserving equivalence. This problem originates from the article where the polynomial hierarchy has been introduced [20], and has been then subject of a number of other studies [1,11,12,19,28]; emphasis is often of Horn formulae. These results differ from the ones in the present article because they are about a different problem. Minimizing a formula means producing an equivalent formula of minimal size; such formula can be syntactically very different from the original one, to the point of not sharing any recognizable part with it. Minimization and redundancy elimination are therefore different tasks, and they serve different purposes: minimization produces a formulae of strictly minimal size; redundancy elimination is not required to reduce size as much as possible, but preserves the structure of the original formula by not adding any new part to it.

The other two classes of related work are about the problems where irredundancy is a restriction over the allowed data and about subproblems of irredundancy. In the first class falls the work of Büning and Zhao [3], who studied problems under the restriction that the involved formulae are irredundant. In the second class we have the problem of minimal unsatisfiability [2,7,21]; this problem can be considered as the subcase of irredundancy when the considered formula is assumed to be unsatisfiable. In this article, we also work in this settings by showing results for 2CNF unsatisfiable formulae.

Liberatore [17] provided complexity results for the CNF case. The present article contains a similar study, but for the 2CNF and Horn case. These special cases are generally considered of importance because consistency and entailment is polynomial, rather than NP-complete, in them. While the Horn case is generally considered the most important of these two subcases, attention to the 2CNF case has also been given, as shown by a number of recent articles on the subject [4,5,13,26,27,29].

The rest of the paper is organized as follows. In the next section, we give a technical overview of the presented results. We then give some general theorems that will be used for proving those results. In the following four sections, results will be shown for the problems of redundancy checking and Irredundant Equivalent Subsets (I.E.S.). In particular, Section 4 contains result about redundancy checking; Section 5 contains the easiest results about I.E.S.'s; Section 6 is about the size of I.E.S.'s; and Section 7 is about the problem of checking whether a clause is in some I.E.S.'s of a given formula. Section 8 contains some results about Horn formulae. Technical lemmas and proofs are moved to the appendix for ease of reading.

## 2. Overview of results

The first problem we consider is that of checking the redundancy of a 2CNF formula. Since a formula is redundant if and only if it is equivalent to one of its subsets and checking equivalence for 2CNF formulae is polynomial, the problem is polynomial. We slightly improve over the trivial algorithm by showing that redundancy can be checked in time $O(nm)$, where $n$ is the number of variables and $m$ is the number of clauses of the formula.

The other problems we consider are about the irredundant equivalent subsets (I.E.S.) of a formula. In particular, the following problems are easily shown to be polynomial for formulae in 2CNF: check whether a formula is an I.E.S. of another one; check whether a clause is in all I.E.S.'s of a formula; and check whether a formula has an unique I.E.S.. The last two problems are polynomial thanks to the following results [17]: a clause $\gamma$ is in all I.E.S.'s of a formula $\Pi$ if and only if $\Pi \backslash \{\gamma\} \not\models \gamma$; a formula $\Pi$ has an unique I.E.S. if and only if $\{\gamma \in \Pi \mid \Pi\backslash\{\gamma\} \not\models \gamma\} \models \Pi$. Combined with the fact that inference is polynomial for 2CNF formulae, these two results imply that checking the presence of a clause in an I.E.S. and checking the uniqueness of I.E.S.'s are polynomial problems.

Two problems about I.E.S.'s require a more complicated complexity analysis: checking whether a clause is in at least an I.E.S. of a formula and checking whether a formula has an I.E.S. of size bounded by an integer $k$. The complexity of these two problems largely depend on the presence of cycles of clauses in the formula. Namely, if the formula contains a cycle of clauses, defined as a sequence of clauses $[\neg l_1 \vee l_2, \neg l_2 \vee l_3, \ldots, \neg l_n \vee l_1]$, these two problems are typically NP-complete, while they are polynomial if the formula does not contain cycles.

The complexity analysis for the 2CNF form has been carried on separately on three different cases. As explained in Section 3.4, equivalence of 2CNF formulae has different features depending on whether the formula is:

(1) inconsistent;
(2) consistent and implying some literals;
(3) consistent and not implying any literal.

We prove that the clauses not containing implied literals and the clauses containing implied literals can be considered separately. More precisely, the problem of redundancy can be solved in two steps:

(1) check the redundancy in $\Pi$ of clauses $l \vee l'$ such that $\Pi \models l$;
(2) remove from $\Pi$ all clauses $l \vee l'$ such that $\Pi \models l$, and check redundancy.

A similar procedure can be used for problems about I.E.S.'s: we indeed prove that every I.E.S. of a consistent formula is composed of two parts, the second being an I.E.S. of the formula composed of the clauses not containing any implied literal. An I.E.S. of a formula can be found by first finding an I.E.S. of this reduced formula and then checking which clauses have to be added to allow the derivation of literals that are implied by the original formula.

While the three conditions of inconsistent formulae, formulae implying literals, and formulae not implying literals require each a different analysis, the complexity is usually the same in all three cases. Namely, the complexity of checking redundancy is always $O(nm)$ regardless of these conditions, while the complexity of the problems of presence in an I.E.S. and of the uniqueness of I.E.S.'s depend more on the presence of cycles in the formula than on the consistency or presence of implied literals.

## 3. Preliminaries

In this paper, we study CNF formulae that are either in 2CNF or in Horn form. Given a set of propositional variables, a literal is a variable or a negated variable. A clause is a disjunction of literals; in particular, a unary/binary clause is a clause composed of one or two literals. A Horn clause is a clause containing at most one positive literal.

A set of clauses containing only unary and binary clauses is a 2CNF formula. A set of clauses composed only of Horn clauses is a Horn formula.

Redundancy of clauses and formulae are defined as follows.

**Definition 1.** A clause $\gamma$ is redundant in a CNF formula $\Pi$ if $\Pi\backslash\{\gamma\} \models \gamma$.

This definition allows a clause $\gamma$ not in $\Pi$ to be classified as redundant in $\Pi$. However, we are typically interested into the redundancy of clauses $\gamma \in \Pi$. Obviously, if $\gamma \in \Pi$ is irredundant in $\Pi$, it is also irredundant in every $\Pi' \subseteq \Pi$.

**Definition 2.** A CNF formula is redundant if it contains a redundant clause.

In this paper we study the problem of checking the redundancy of 2CNF and Horn formulae, and some problems related to making a formula irredundant by eliminating some redundant clauses. What results from this process is formalized by the following definition.

**Definition 3.** (See [17].) An Irredundant Equivalent Subset (I.E.S.) of a CNF formula $\Pi$ is a formula $\Pi'$ such that $\Pi' \subseteq \Pi$, $\Pi' \equiv \Pi$, and $\Pi'$ is irredundant.

Every formula has at least one I.E.S. An irredundant formula has a single I.E.S., which is the formula itself. A redundant formula can have a number of I.E.S.'s ranging from one to exponentially many [17]. The following properties have been proved in a previous article.

**Property 1.** *(See [17].) A clause $\gamma$ is in all* I.E.S.*'s of a formula $\Pi$ if and only if $\gamma$ is irredundant in $\Pi$.*

**Property 2.** *(See [17].) A formula $\Pi$ has a single* I.E.S. *if and only if $\{\gamma \in \Pi \mid \Pi\backslash\{\gamma\} \not\models \gamma\}\} \models \Pi$.*

For reasons that will be explained shortly, the set of literals that are entailed by a formula is technically important while studying redundancy. The following notation is used for the literals that are entailed by a formula.

**Notation.** $\Pi_{\models} = \{l \mid \Pi \models l\}$.

The following notation for the clauses containing a literal in a set will be used.

**Notation.** $\Pi|\{l_1, \ldots, l_m\} = \{\gamma \mid l_i \in \gamma,\ \gamma \in \Pi\}$.
We also use $\Pi|l = \Pi|\{l\}$, where $l$ is a literal.

## 3.1. Unit propagation

The proofs of complexity of redundancy of 2CNF formulae are mostly done using unit propagation. The following lemmas, whose proofs can be found in the appendix, show how entailment is related to unit propagation. We denote by $\models_R$ the derivation by resolution and by $\models_{UP}$ the derivation by unit propagation.

In what follows, $\Pi$ denotes a 2CNF formula, i.e., a set of clauses, each composed at most of two literals. We assume that $\Pi$ does not contain the empty clause. By resolution trees we mean regular resolution trees; their root is labeled with a clause which is not necessarily $\bot$. The following property immediately follows from resolution being a complete inference method for prime implicates [22].

**Property 3.** *For any set of clauses $\Pi$ and clause $\gamma$, it holds $\Pi \models \gamma$ if and only if there exists $\gamma' \subseteq \gamma$ such that $\Pi \models_R \gamma'$.*

When applied to binary clauses, this property can be reformulated as:

$\Pi \models l_1 \vee l_2$ if and only if one of the following conditions hold: $\Pi \models_R \bot$
$\Pi \models_R l_1$
$\Pi \models_R l_2$
$\Pi \models_R l_1 \vee l_2$

Each of these three cases (inconsistency, entailment of at least one literal, entailment of the clause only), corresponds to a property of unit resolution, as the following lemmas (proved in the appendix) show.

**Lemma 1.** *For any 2CNF formula $\Pi$ and two literals $l_1$ and $l_2$, if $\Pi \models_R l_1 \vee l_2$, then $\Pi \cup \{\neg l_1\} \models_{UP} l_2$.*

**Lemma 2.** *For any 2CNF formula $\Pi$ and literal $l$, if $\Pi \models_R l$ then $\Pi \cup \{\neg l\} \models_{UP} \bot$.*

**Lemma 3.** *A 2CNF formula $\Pi$ is inconsistent if and only if there exists a variable $x$ such that $\Pi \cup \{x\} \models_{UP} \bot$ and $\Pi \cup \{\neg x\} \models_{UP} \bot$.*

These lemmas have the following easy consequence.

**Lemma 4.** *For every 2CNF formula $\Pi$ and literals $l_1$ and $l_2$, $\Pi \models l_1 \vee l_2$ if and only if $\Pi$ is inconsistent or $\Pi \cup \{\neg l_1\} \models_{UP} \bot$ or $\Pi \cup \{\neg l_2\} \models_{UP} \bot$ or $\Pi \cup \{\neg l_1\} \models_{UP} l_2$.*

## *3.2. Unit clauses*

If $\Pi$ only contains binary clauses (no unary clause), then $\Pi \cup \{l\} \models_{UP} l'$ means that $l'$ is obtained by propagating $l$ in $\Pi$, as $l$ is the only unit clause of $\Pi \cup \{l\}$. If this is the case, $\Pi \cup \{l\} \models_{UP} l'$ is equivalent to the reachability of $l'$ from $l$ in the graph of $\Pi$ induced by $l$.

We generally do not assume that $\Pi$ is composed of binary clauses only. In particular, even if we start from a formula made of binary clauses only, applying Corollary 1 or Lemma 3 leads to formulae containing unit clauses. On the other hand, every unit clause $l$ can be replaced with the logically equivalent pair $\{l \vee x, l \vee \neg x\}$, where $x$ is a new variable not occurring in the rest of the formula. Since $\{l\} \equiv \{l \vee x, l \vee \neg x\}$, the redundancy of $l$ is equivalent to the redundancy of the pair $\{l \vee x, l \vee \neg x\}$, and most of the properties related to I.E.S.'s are also unaffected by this replacement.

The only property that is changed by replacing $l$ with $\{l \vee x, l \vee \neg x\}$ is about the size of I.E.S.'s of a formula, as we are replacing a single clause with a pair of clauses. This problem will be taken care by counting such a pair as if it were a single clause in the algorithms for checking the size of a minimal I.E.S. of a formula.

In the rest of this paper, whenever we have to check whether $\Pi \cup \{l_1\} \models_{UP} l_2$ or $\Pi \cup \{l_1\} \models_{UP} \bot$, we assume that this transformation has been preliminary done on $\Pi$, so that $\Pi$ only contains binary clauses. This way, checking unit propagation can be done by looking at the graph of $\Pi$ induced by $l_1$. In particular, $\Pi \cup \{l_1\} \models_{UP} l_2$ means that, in the graph of $\Pi$ induced by $l_1$, there is a path from $l_1$ to $l_2$.

By definition, $\Pi \cup \{l_1\} \models_{UP} \bot$ means that unit propagation from $l$ allows reaching a pair of opposite literals $l_2$ and $\neg l_2$. Graphically, there exists a path from $l_1$ to $l_2$ and a path from $l_1$ to $\neg l_2$ in the graph of $\Pi$ induced by $l_1$. Let $l_3$ be the last common literal of these two paths.

Graphically, this condition can be represented as in Fig. 1. Since $\neg l_2$ can be reached from $l_3$, we have that $\neg l_3$ can be reached from $l_2$. As a result, there exists a path from $l_1$ to $l_3$, from $l_3$ to $l_2$, and from $l_2$ to $\neg l_3$. In other words,

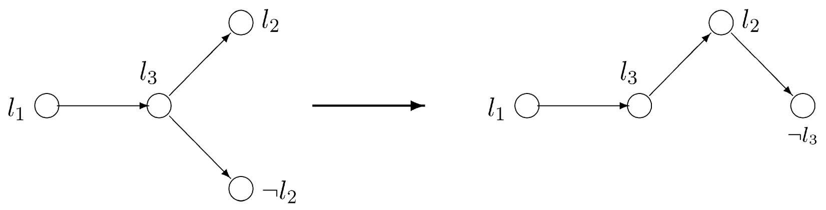

Fig. 1. Two ways of viewing a contradiction entailed by a literal.

$\Pi \cup \{l_1\} \models_{UP} \bot$ implies that there exists a *single* path starting from $l_1$ that includes a pair of opposite literals. In the sequel, whenever we say "all clauses in a path from $l_1$ to $\bot$", we mean a single path starting from $l_1$ and ending with *the first* literal that is the opposite of another literal in the path.

## 3.3. Cyclicity and induced graphs

The presence of cycles of clauses in a formula determines the complexity of some problems related to I.E.S.'s. Formally, cycles are defined as follows.

**Definition 4** *(Simple cycle of binary clauses).* A cycle of binary clauses is a sequence of clauses $[\neg l_1 \vee l_2, \neg l_2 \vee l_3, \ldots, \neg l_n \vee l_1]$ such that no literal $l_i$ occurs in more than one clauses (no variable occurs in more than two).

This definition only covers simple cycles of clauses, that is, we are not allowed to "cross" the same literal twice. Defining a non-simple cycle of clauses is possible but not necessary in this paper because we only classify formulae depending on whether they have cycles or not.

The graph of a 2CNF formula induced by a literal is, roughly speaking, the graph of literals that can be derived from the given one by unit propagation.

**Definition 5.** The graph induced by a literal $l$ on a 2CNF formula $\Pi$ is the minimal (as of containment of the sets of nodes and edges) graph $(N, E)$ such that:

(1) $l \in N$;
(2) if $l' \in N$ and $\neg l' \vee l'' \in \Pi$, then $l'' \in N$ and $(l', l'') \in E$.

A property of acyclic formulae is that all their induced subgraphs are acyclic and vice versa.

**Property 4.** *A 2CNF formula $\Pi$ contains simple cycles if and only if some of its induced graphs contain cycles.*

Formulae not containing cycles have some interesting properties. First, consistent acyclic 2CNF formulae not entailing any literal have a single I.E.S., which makes some problems related to I.E.S.'s computationally simpler. The second result about acyclic formulae is that, for every two literals $l_1$ and $l_2$ that are entailed by the formula, the choice of a minimal subset of clauses entailing $l_1$ is independent on the choice for $l_2$. These two results are proved in two later sections.

The following lemma shows that a chain of clauses that prove that $\Pi \cup \{l\} \models_{UP} \bot$ cannot share clauses if the formula is acyclic.

**Lemma 5.** *In an acyclic 2CNF formula, the two chains of clauses from $l$ to $l'$ and from $l'$ to $\neg l'$ do not share clauses.*

**Proof.** If both chains contain the same clause $l'' \vee l'''$, then the formula is cyclic, as these two chains cross the same literal $l''$ twice. If the first chain contains $l'' \vee l'''$ and the second $l''' \vee l''$, then a cycle of clauses is $l''' \vee l''$, the chain from $l''$ to $\neg l'$, and the reverse of the chain from $l'''$ to $l''$. □

## 3.4. Analysis is done on three cases separately

As already mentioned, the analysis of the problem of redundancy checking and of the problems about I.E.S.'s is done separately for three cases:

(1) the formula is inconsistent;
(2) the formula is consistent and implies some literals;
(3) the formula is consistent and does not imply literals.

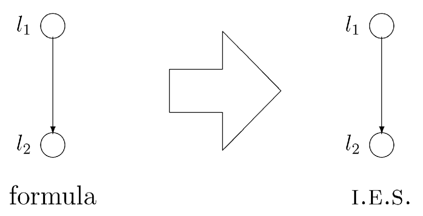

Fig. 2. An I.E.S. of a consistent formula.

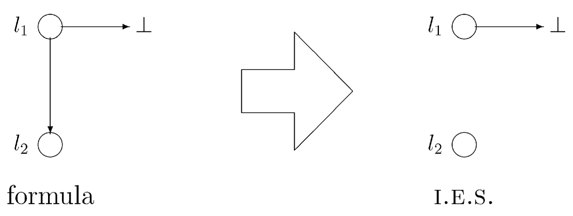

Fig. 3. An I.E.S. of a consistent formula entailing a literal.

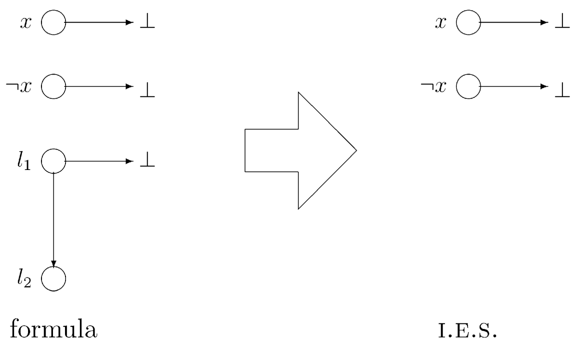

Fig. 4. An I.E.S. of an inconsistent formula.

We can now explain why these three cases require a different analysis. Let us consider first the last case: a formula not implying any literal. By Lemma 4, a clause is entailed by a formula if and only if one of its literals can be derived by unit propagation from the negation of the other one. If $\gamma \in \Pi$ and $\Pi'$ is an I.E.S. of $\Pi$, then $\Pi' \models \gamma$. As a result, the problem of redundancy and the problems about I.E.S.'s can be reformulated in terms of formulae whose induced graphs have the same reachability relation of the graphs of the original formula. Fig. 2 exemplifies this idea.

This requirement, however, is too restrictive when considering literals that are implied by the formula. For example, if $l_2$ is reachable from $l_1$ in the original formula but $\neg l_1$ is entailed by the formula, this reachability condition is not necessarily true in all I.E.S.'s of the formula. Indeed, all that is needed is that unit propagation from $l_1$ allows reaching contradiction; the condition that $l_2$ is reachable from $l_1$ is not necessarily true in all I.E.S.'s of the formula. Such a condition is shown in Fig. 3.

Finally, if a formula is inconsistent then its I.E.S.'s are only required to be inconsistent. Not only these I.E.S.'s can have a reachability relation different from that of the original formula; they can even omit to mention some literals at all. Indeed, a subset of an inconsistent formula can be inconsistent even if it does not mention some literals of the original formula. Fig. 4 shows an example where an I.E.S. does not mention two variables $l_1$ and $l_2$ of the original formula, and consistency is guaranteed via variable $x$.

Summarizing, the three cases are studied separately because of the different requirements on equivalent subsets: in the first case, reachability using unit propagation is the same in the I.E.S.'s and in the formula; in the second case, only reachability of $\bot$ from the negation of implied literals is the same; in the third case, only the reachability of $\bot$ from an arbitrary pair of opposite literals is the same.

### *3.5. Inconsistent 2CNF formulae*

In this section we present some results about irredundant inconsistent 2CNF formulae. The first result is that such a formula cannot be acyclic, if it contains no unary clauses but only binary ones.

**Theorem 1.** *Every inconsistent set of binary clauses is cyclic.*

In general, a 2CNF may contain unary clauses, and can therefore be cyclic and inconsistent at the same time. As an example, $\{x, \neg x \vee y, \neg x \vee \neg y\}$ is inconsistent but acyclic. If a 2CNF formula includes unary clauses, we generally turn these clauses into equivalent binary clauses: $x$ is for example replaced by $\neg x \vee z$ and $\neg x \vee \neg z$, where $z$ is a new variable. This transformation can create cycles but, as proved by the next lemma, this only happens if the formula is inconsistent.

**Lemma 6.** *Every 2CNF formula $\Pi$ containing a unit clauses $l$ is equivalent to $\Pi \backslash \{l\} \cup \{l \vee y, l \vee \neg y\}$, where $y$ is a new variable; this second formula is cyclic only if $\Pi$ is either cyclic or inconsistent.*

An acyclic 2CNF can be inconsistent only if it contains unary clauses. Since its I.E.S.'s are inconsistent as well, they contain at least one unary clause each. On the other hand, an irredundant acyclic 2CNF formula contains at most two unary clauses. This is proved using the following lemma.

**Lemma 7.** *Every resolution tree rooted with $\perp$ of a 2CNF formula has at most two unary clauses in the leaves.*

This lemma proves that inconsistency of a 2CNF formula can always be proved using a resolution tree having at most two unary clauses in the leaves. Since the set of leaves of such tree are sufficient to prove inconsistency, two unary clauses are always sufficient to prove inconsistency.

**Lemma 8.** *Every irredundant inconsistent 2CNF formula contains at most two unary clauses.*

The following theorem shows that, if the binary clauses of a 2CNF formula are consistent while the whole formula is inconsistent, this inconsistency can be proved using unit propagation.

**Lemma 9.** *If $\Pi$ is an irredundant inconsistent 2CNF, $\Pi'$ and $\Pi''$ are the set of its unary and binary clauses, respectively, and $\Pi''$ is consistent, then $\Pi' \cup \Pi'' \models_{UP} \perp$.*

These lemmas, together with Lemma 5, will be used to prove that the considered problems about I.E.S.'s are polynomial on 2CNF formulae that are both inconsistent and irredundant. This is because one can test propagation from one unit clause at time, checking the minimal number of clauses needed to reach, by unit propagation, either a pair of opposite literals or the negation of another unit clause.

## 3.6. Formulae implying literals

A consistent 2CNF formula $\Pi$ may imply some literals or not. We show that, as long as redundancy and I.E.S.'s are concerned, we can treat the clauses containing an implied literal and those not containing them separately. A first obvious property is that we can replace all clauses containing an entailed literal with the literal itself: if $\Pi \models l$, then we can add $l$ to $\Pi$ and remove all clauses of the form $l \vee l'$.

**Lemma 10.** *If $\Pi$ is a CNF formula such that $\Pi \models l$, then $\Pi$ and $\Pi \backslash (\Pi | l) \cup \{l\}$ are equivalent.*

Replacing all clauses containing an entailed literal with the literal itself does not only preserve equivalence but also the redundancy of clauses not containing the entailed literal.

**Lemma 11.** *Let $\Pi$ be a formula implying the literal $l$. If the clause $\gamma \in \Pi$ does not contain $l$, then $\gamma$ is redundant in $\Pi$ if and only if it is redundant in $\Pi \backslash (\Pi | l) \cup \{l\}$.*

This lemma shows that clauses not containing literals entailed by the formula can be checked for redundancy after the clauses containing entailed literals have been replaced by the entailed literals. The following corollary is an obvious consequence of this lemma.

**Corollary 1.** *If $\Pi$ is a consistent 2CNF such that $\Pi \models l$, then $\Pi$ contains a redundant clause not containing $l$ if and only if $\Pi \backslash (\Pi | l) \cup \{l\}$ is redundant.*

This corollary shows that, once we have determined that all clauses containing a literal $l$ such that $\Pi \models l$ are irredundant, we can replace all such clauses with $l$. Repeating this procedure for all literals implied by the formula, we obtain a formula with disjoint unit and binary clauses. This is because, if $\Pi \models l$, every clause $l \vee l'$ is replaced by $l$, while every clause $\neg l \vee l'$ is replaced by $l'$ as $\Pi \models l'$. As a result, no binary clause contains a variable that is in a unit clause in the resulting formula.

A similar result holds about I.E.S.'s: replacing clauses containing entailed literals with these literals both in the formula and in one of its I.E.S.'s generates a formula and one of its I.E.S.'s.

**Lemma 12.** *Let $\Pi$ be a consistent 2CNF such that $\Pi \models l$. If $\Pi'$ is an* I.E.S. *of $\Pi$ then $\Pi_2 = \Pi' \backslash (\Pi|l) \cup \{l\}$ is an* I.E.S. *of $\Pi \backslash (\Pi|l) \cup \{l\}$.*

The converse of this lemma does not hold. Even if $\Pi_2$ is an I.E.S. of $\Pi \backslash (\Pi|l) \cup \{l\}$, it is not necessarily true that an I.E.S. of $\Pi$ can be obtained by simply adding some clauses of $\Pi|l$ to it. Actually, even adding *all* clauses of $\Pi|l$ does not necessarily lead to a formula that is equivalent to $\Pi$, as the following example shows.

$$\Pi = \{x \vee x_1, x \vee x_2, \neg x_1 \vee y, \neg x_2 \vee \neg y, \neg x \vee y\}$$
$$l = x$$
$$\Pi \backslash (\Pi|l) \cup \{l\} = \{\neg x_1 \vee y, \neg x_2 \vee \neg y, \neg x \vee y, x\}$$
$$\Pi_2 = \{\neg x_2 \vee \neg y, \neg x \vee y, x\}$$
$$\Pi_2 \backslash \{l\} \cup (\Pi|l) = \{\neg x_2 \vee \neg y, \neg x \vee y, x \vee x_1, x \vee x_2\}$$

It holds $\Pi \models x$: this can be proved by adding $\neg x$ to $\Pi$ and using unit propagation: $x_1$ and $x_2$ are derived, leading to $y$ and $\neg y$, respectively. It is also easy to prove that $\Pi_2$ is equivalent to $\Pi \backslash (\Pi|l) \cup \{l\}$: since $\neg x \vee y$ and $x$ entail $y$, the clause $\neg x_1 \vee y$ is entailed by $\Pi_2$. The irredundancy of $\Pi_2$ is also easy to prove: removing a single clause from it, either $x$, $y$, or $\neg x_2$ cannot be derived any longer. Adding all clauses of $\Pi|l$ to $\Pi_2$, however, does not allow to derive $x$ any longer. Indeed, $\Pi_2 \backslash \{l\} \cup (\Pi|l)$ has the model in which $\neg x$, $x_1$, $x_2$, and $\neg y$ are true, which assigns false to $x$.

An analysis of this counterexample shows why the converse of Lemma 12 is false. The problem is with the clause $\neg x \vee y$, which makes $y$ true in the original formula. This clause allows to remove $\neg x_1 \vee y$, which was necessary in $\Pi$ to entail $x$. Without the clause $\neg x \vee y$, indeed, the converse of Lemma 12 would hold for $\Pi$ and $l = x$. In the appendix we detail how this counterexample invalidates a candidate proof of the converse of Lemma 12.

The following easy lemma and corollary show a separation in the variables of some clauses.

**Lemma 13.** *Every clause of a 2CNF formula $\Pi$ either is in $\Pi|\Pi_{\models}$ or does not contain literals in $\Pi_{\models}$ nor their negation.*

**Corollary 2.** *For every 2CNF formula $\Pi$, it holds that $\Pi \backslash (\Pi|\Pi_{\models})$ and $\Pi_{\models}$ do not share variables.*

This corollary is the base of the next result. Indeed, it shows that $\Pi \backslash (\Pi|\Pi_{\models}) \cup \Pi_{\models}$ is composed of two parts $\Pi \backslash (\Pi|\Pi_{\models})$ and $\Pi_{\models}$ not sharing variables. The same result therefore holds when referred to an arbitrary subset of $\Pi$, and in particular it holds for all of its I.E.S.'s. The following lemma proves that replacing *at once* all clauses of $\Pi|\Pi_{\models}$ with $\Pi_{\models}$, the converse of Lemma 12 holds.

**Lemma 14.** *Let $\Pi$ be a consistent 2CNF formula. If $\Pi_2$ is an* I.E.S. *of $\Pi \backslash (\Pi|\Pi_{\models}) \cup \Pi_{\models}$, then $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})$ is equivalent to $\Pi$.*

**Proof sketch.** Since $\Pi$ and $\Pi \backslash (\Pi|\Pi_{\models}) \cup \Pi_{\models}$ are equivalent and $\Pi_2$ is an I.E.S. of $\Pi \backslash (\Pi|\Pi_{\models}) \cup \Pi_{\models}$, we have that $\Pi_2$ and $\Pi$ are equivalent. The claim is proved by showing that $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})$ entails $\Pi_{\models}$. This would prove that $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})$ is equivalent to $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models}) \cup \Pi_{\models}$, which is a superset of $\Pi_2$ and is therefore equivalent to $\Pi$.

Intuitively, the proof is as follows: if $l \in \Pi_{\models}$, then there is a proof of $l$ in $\Pi$. This proof involves some clauses of $\Pi|\Pi_{\models}$ and some clauses in $\Pi \backslash (\Pi|\Pi_{\models})$. On the other hand, everything that is entailed in $\Pi$ is also entailed in its equivalent formula $\Pi_2 \backslash (\Pi | \Pi_{\models}) \cup \Pi_{\models}$. Since $\Pi_2 \backslash (\Pi | \Pi_{\models})$ and $\Pi_{\models}$ are built over disjoint literals, every clause of $\Pi$ not containing literals that are entailed by $\Pi$ is derivable in $\Pi_2 \backslash (\Pi | \Pi_{\models})$. □

This lemma only proves that $\Pi_2 \backslash \Pi_{\models} \cup (\Pi | \Pi_{\models})$ is equivalent to $\Pi$, but does not prove it is an I.E.S. of $\Pi$. In general, this is not true. However, we can show that an I.E.S. can be obtained from this set by removing only some clauses of $\Pi | \Pi_{\models}$.

**Lemma 15.** *Let $\Pi$ be a consistent 2CNF formula. If $\Pi_2$ is an* I.E.S. *of $\Pi \backslash (\Pi | \Pi_{\models}) \cup \Pi_{\models}$, then there exists $\Pi_1 \subseteq \Pi | \Pi_{\models}$ such that $\Pi_1 \cup (\Pi_2 \backslash \Pi_{\models})$ is an* I.E.S. *of $\Pi$.*

**Proof sketch.** Lemma 14 shows that adding $\Pi | \Pi_{\models}$ to $\Pi_2 \backslash \Pi_{\models}$ results in a set that is equivalent to $\Pi$. The statement of the present lemma could be false only if an I.E.S. could be obtained by removing a clause of $\Pi_2 \backslash \Pi_{\models}$. This is equivalent to the redundancy of such a clause in $\Pi_2 \backslash \Pi_{\models} \cup (\Pi | \Pi_{\models})$. This is in turn proved to be false by showing, via some simple manipulation of formulae, that it is equivalent to the redundancy of $\Pi_2$. □

The converse of this lemma is an immediate consequence of a repeated application of Lemma 12. We can therefore conclude the following corollary.

**Corollary 3.** *$\Pi_2$ is an* I.E.S. *of $\Pi \backslash (\Pi | \Pi_{\models}) \cup \Pi_{\models}$ if and only if there exists $\Pi_1 \subseteq \Pi | \Pi_{\models}$ such that $\Pi_1 \cup (\Pi_2 \backslash \Pi_{\models})$ is an* I.E.S. *of $\Pi$.*

We show a lemma about clauses containing a literal that is entailed by the formula.

**Lemma 16.** *Every consistent 2CNF formula $\Pi$ implying a literal $l$ and containing three or more clauses containing $l$ contains at least a redundant clause containing $l$.*

**Proof sketch.** If $l \in \Pi$, any other clause containing $l$ is redundant. If $l \notin \Pi$, the proof is based on the fact that $\Pi \cup \{\neg l\} \models_{UP} \bot$ since $\Pi$ is consistent. In turn, this is equivalent to $\neg l$ implying two opposite literals by unit propagation, which requires two (possibly disjoint) chains of clauses. The first two clauses of these chains are therefore the only two clauses containing $l$ that are necessary to entail $l$; all other clauses containing $l$ are then entailed by $l$ and therefore redundant. □

An obvious consequence of this lemma is that every consistent and irredundant 2CNF formula implying a literal contains at most two clauses containing that literal.

## *3.7. Acyclic consistent 2CNF formulae not implying literals have a single* I.E.S.

We show that every acyclic consistent 2CNF formula not implying any literal has a single I.E.S. If $\Pi$ is consistent and $\Pi \not\models l$ for every literal $l$, we have that $\Pi \models \neg l \vee l'$ holds if and only if $\Pi \cup \{l\} \models_{UP} l'$. The same holds for all subsets of $\Pi$ and, in particular, for all its I.E.S.'s.

**Lemma 17.** *If $\Pi$ is a consistent acyclic 2CNF and $\Pi \cup \{l\} \models_{UP} l'$, then $\Pi' \cup \{l\} \models_{UP} l'$ holds for every* I.E.S. *$\Pi'$ of $\Pi$.*

**Proof sketch.** Both antecedent and consequent of the statement are equivalent to entailment of $l \vee l'$, and a formula and its I.E.S.'s are equivalent and therefore imply the same clauses. □

Every literal $l$ partitions $\Pi$ into the set of clauses that are involved in the first step of unit propagation from $l$ and the other ones:

$$D_\Pi(l) = \{\gamma \in \Pi \mid \gamma = \neg l \vee l'\}$$
$$R_\Pi(l) = \Pi \backslash D_\Pi(l)$$

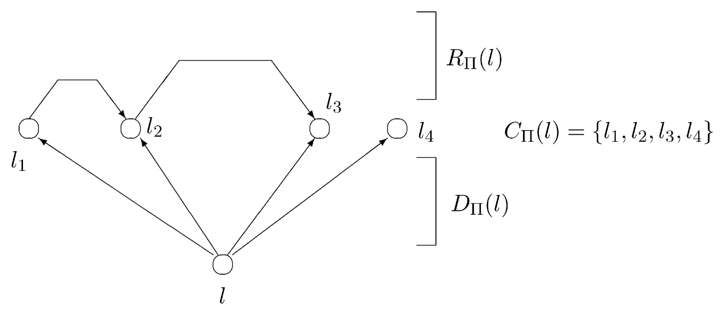

Fig. 5. The graph induced by a literal $l$ on a 2CNF formula.

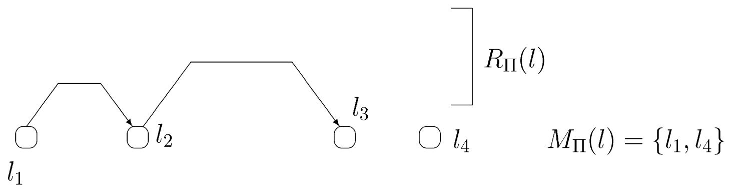

Fig. 6. $R_\Pi(l)$ and $M_\Pi(l)$.

The clauses in $D_\Pi(l)$ are those used in the first step of unit propagation from $l$. The following set $C_\Pi(l)$ is the set of literals that would result from this propagation.

$$C_\Pi(l) = \{l' \mid \neg l \vee l' \in \Pi\} = \{l' \mid \neg l \vee l' \in D_\Pi(l)\}$$

Since all clauses $\neg l \vee l'$ in $D_\Pi(l)$ are also in $\Pi$, they are entailed by every I.E.S. $\Pi'$ of $\Pi$. Since $\Pi$ is a consistent CNF, so are all its subsets, and $\Pi'$ in particular. Since $\Pi'$ entails $\neg l \vee l'$ and $\Pi'$ is consistent and not entailing literals, $l'$ is reachable from $l$ in the graph induced by $l$ on $\Pi'$.

In turn, a given $l' \in C_\Pi(l)$ is reachable from $l$ if and only if either $\neg l \vee l' \in \Pi'$, or there is another literal $l''$ such that $\neg l \vee l'' \in \Pi'$ and $l'$ is reachable from $l''$ using the edges corresponding to the clauses of $R_\Pi(l)$.

Fig. 5 shows an example of a graph induced by a literal $l$ on a 2CNF formula. In this example, $\neg l \vee l_4$ cannot be removed from $\Pi$ because it is the only clause allowing $l_4$ to be reached from $l$. The same holds for $\neg l \vee l_1$. The clauses $\neg l \vee l_2$ and $\neg l \vee l_3$ are instead redundant because $l_2$ and $l_3$ can be reached from $l$ by following the edge corresponding to the clause $\neg l \vee l_1$ and then following some edges corresponding to clauses in $R_\Pi(l)$.

This example shows that the irredundant clauses are those containing the literals of $C_\Pi(l)$ that cannot be reached from other literals of $C_\Pi(l)$. Such literals necessarily exist because the formula (and therefore all its induced graphs) is acyclic (and finite).

$$M_\Pi(l) = \{l' \in C_\Pi(l) \mid \nexists l'' \in C_\Pi(l) \text{ such that } R_\Pi(l) \cup \{l''\} \models_{UP} l'\}$$

Formally, $M_\Pi(l)$ is the set of literals that cannot be reached from other literals of $C_\Pi(l)$ using unit propagation on the clauses $R_\Pi(l)$. In the example above, $R_\Pi(l)$ and $M_\Pi(l)$ are as shown in Fig. 6.

The nodes that cannot be reached from other nodes are $l_1$ and $l_4$. Therefore, $M_\Pi(l) = \{l_1, l_4\}$. The following corollary states that $M_\Pi(l)$ characterizes the irredundant clauses containing $l$. Its ancillary lemmas and their proofs are in the appendix.

**Corollary 4.** *A clause $\neg l \vee l_1$ is in an* I.E.S. *of the consistent acyclic 2CNF formula $\Pi$ if and only if $l_1 \in M_\Pi(l)$.*

Since the statement of this corollary is in an "if and only if" form, once $l_1 \vee l_2$ is proved to be in an I.E.S. because $l_2 \in M_\Pi(\neg l_1)$, we do not need to also check $l_1 \in M_\Pi(\neg l_2)$. The same holds if $l_1 \vee l_2$ is proved not to be in an I.E.S. in the same way. This result tells that every 2CNF consistent acyclic formula that does not imply literals has a single I.E.S.

**Theorem 2.** *Every consistent acyclic 2CNF formula $\Pi$ not implying literals has a single* I.E.S.

**Proof.** For each clause $l_1 \vee l_2$, check whether $l_2 \in M_\Pi(\neg l_1)$. If this is true, then $l_1 \vee l_2$ is in all I.E.S.'s, otherwise it is in no I.E.S. Therefore, the set composed of all clauses $l_1 \vee l_2$ such that $l_2 \in M_\Pi(\neg l_1)$ is the single I.E.S. of $\Pi$. □

## *3.8. Implied literals not in a cycle of a consistent 2CNF formula*

In this section, we consider the redundancy of clauses containing a literal that is entailed by a consistent formula but is not in a cycle of clauses. We show that, if $l'$ and $l''$ are two such literals, then we can independently choose among clauses containing $l'$ and from clauses containing $l''$ to form an I.E.S. In other words, an I.E.S. can be obtained by choosing a subset of clauses containing $l'$ and a subset of clauses containing $l''$, and these choices are independent from each other.

By Lemma 2, since $\Pi$ is consistent, $\Pi \models \neg l$ holds if and only if $\Pi \cup \{l\} \models_{UP} \bot$. Using the transformation of Section 3.2, we can assume that $\Pi$ does not contain unary clauses. Therefore, $\Pi \cup \{l\} \models_{UP} \bot$ means that we can reach a pair of opposite literals by propagating $l$ in $\Pi$. The following notations have been introduced in the previous section:

$$D_\Pi(l) = \{\gamma \in \Pi \mid \gamma = \neg l \vee l'\}$$
$$R_\Pi(l) = \Pi \backslash D_\Pi(l)$$
$$C_\Pi(l) = \{l' \mid \neg l \vee l' \in \Pi\}$$

Since $\Pi \models \neg l$, every equivalent subset of $\Pi$ allows reaching a pair of opposite literals from $l$. This is possible if and only if either one of the two following conditions is true:

(1) $\neg l \vee l_1 \in \Pi$ and $\Pi \cup \{l_1\} \models_{UP} \bot$;
(2) $\neg l \vee l_2 \in \Pi$ and $\Pi \cup \{l_2\} \models_{UP} \neg l$, which is equivalent to:
$\neg l \vee l_2 \in \Pi$, $\Pi \cup \{l_2\} \models_{UP} \neg l_3$, and $l_3 \vee \neg l \in \Pi$.

If $\Pi'$ is an I.E.S. of $\Pi$, these conditions hold for $\Pi$ if and only if they hold for $\Pi'$. In addition, if $\Pi$ does not contain a cycle including $l$, the same holds for $\Pi'$. As a result, unit propagation from $l_1$ to $\bot$ or from $l_2$ to $\neg l_3$ cannot include $l$, as otherwise $l$ would be part of a cycle.

Consider the formula graphically represented in Fig. 7. All paths from $l_1$ to $\bot$ and from $l_2$ to $\neg l_3$ are entirely contained in $R_\Pi(l)$: otherwise, $l$ would be part of a cycle. As a result, the same derivations are possible in an I.E.S. $\Pi'$ of $\Pi$ if and only if they are possible using only clauses of $R_\Pi(l)$, that is, they are possible in $\Pi' \cap R_\Pi(l)$. In other words, $\Pi' \cup \{l_1\} \models_{UP} \bot$ if and only if $(\Pi' \cap R_\Pi(l)) \cup \{l_1\} \models_{UP} \bot$, and the same for the derivation of $\neg l_3$ from $l_2$. As a result, $\Pi' \cap R_\Pi(l)$ always entails $l$ with the addition of either $\neg l \vee l_1$ or the pair $\neg l \vee l_2$ and $\neg l \vee l_3$.

Let us formally denote the sets of literals like $l_1$ and of pairs of literals like $l_2$ and $l_3$.

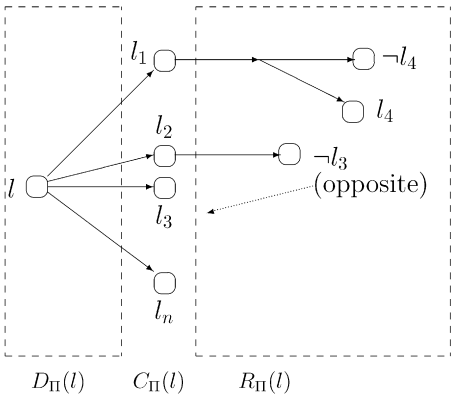

Fig. 7. An example formula.

$$S_\Pi(l) = \{l_1 \in C_\Pi(l) \mid R_\Pi(l) \cup \{l_1\} \models_{UP} \bot\}$$
$$P_\Pi(l) = \{(l_2, l_3) \in C_\Pi(l) \mid R_\Pi(l) \cup \{l_2\} \models_{UP} \neg l_3\} \backslash S_\Pi(l)$$

In words, $S_\Pi(l)$ is the set of literals we can reach contradiction from using unit propagation in $R_\Pi(l)$ while $P_\Pi(l)$ is composed of pairs of literals such that the negation of one is reachable from the other in $R_\Pi(l)$. The above reasoning is formalized by the next lemma, which is formally proved in the appendix.

**Lemma 18.** *If $\Pi'$ is an* I.E.S. *of a consistent acyclic 2CNF formula $\Pi$ such that $\Pi \models \neg l$ and $l$ is not in any cycle of clauses of $\Pi$, then the following are all* I.E.S.*'s of $\Pi$:*

(1) $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_1\}$ *with* $l_1 \in S_\Pi(l)$;
(2) $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_2, \neg l \vee l_3\}$ *with* $(l_2, l_3) \in P_\Pi(l)$.

This lemma shows a simple way for determining an I.E.S. of an acyclic consistent formula: for each literal $l$ such that $\Pi \models \neg l$, we choose either a clause $l \vee l_1$ with $l_1 \in S_\Pi(l)$ or a pair of clauses $l \vee l_2$ and $l \vee l_3$ with $(l_2, l_3) \in P_\Pi(l)$. By the above lemma, we can do this choice for all clauses containing a literal that is entailed by the formula.

## 4. Redundancy checking

In this section, we consider the problem of checking whether a 2CNF formula is redundant. If $m$ is the number of clauses of a 2CNF formula, the size such a formula is between $m$ and $2m$. Checking consistency of a set of binary clauses can be done in linear time, and therefore in time $O(m)$. The trivial algorithm of checking whether $\Pi \backslash \{\gamma\} \models \gamma$ for each $\gamma \in \Pi$ requires $O(m)$ for each clause, and therefore $O(m^2)$ total time. We improve over this result by showing algorithms that solve the problem in time $O(nm)$, where $n$ is the number of variables of the formula, in all three possible cases (inconsistent formulae, formulae implying literals or not.) Cyclicity does not affect the problem of checking redundancy. Here is a summary of the results in the various cases.

*The formula is inconsistent.* If a set is both inconsistent and irredundant, the number of its clauses is at most four times the number of variables. Therefore, if the number of clauses is greater, the set is redundant. If it is lower, we still have to check redundancy, but the running time $O(mm)$ of the trivial algorithm is now the same as $O(nm)$.

*The formula is consistent.* For each literal $l$, we proceed differently depending on whether $\Pi \models l$ holds or not.

*The formula implies the literal.* If $\Pi \models l$ then either $\Pi$ is inconsistent or $\Pi \models_R l$. Since $\Pi$ is by assumption consistent, we have $\Pi \models_R l$. By Lemma 2, we have $\Pi \cup \{\neg l\} \models_{UP} \bot$. We consider two cases separately.

*$l$ is in three or more clauses.* Formula $\Pi$ is redundant by Lemma 16.

*$l$ is in one or two clauses.* By assumption $\Pi \cup \{\neg l\} \models_{UP} \bot$. In order to check redundancy, just remove any of the two clauses containing $l$ and check whether UP still leads to $\bot$ from $l$. Two UP derivations, which are linear in the number of clauses, are needed for the literal $l$.

*The formula does not imply the literal.* Since $\Pi \not\models l$ we have $\Pi \backslash \{\neg l \vee l'\} \models \neg l \vee l'$ if and only if $\Pi \backslash \{\neg l \vee l'\} \cup \{l\} \models_{UP} l'$. Therefore, we can consider the graph induced by the unit propagation of $l$ in $\Pi$, and check whether $l'$ is reachable from $l$ without using the edge corresponding to $\neg l \vee l'$. This test can be done in linear time by a modified algorithm of graph reachability.

The description of the algorithms and their proofs of correctness are given in the following three sections.

### 4.1. Redundancy checking: inconsistent 2CNF formulae

Lemma 3 shows that every inconsistent 2CNF formula contains some clauses allowing both $x$ and $\neg x$ to derive $\bot$ by unit propagation. In turn, this implies the existence of two acyclic chains of clauses that allow the derivation of two opposite literals from $x$ and $\neg x$, respectively.

**Lemma 19.** *A 2CNF formula $\Pi$ is inconsistent and irredundant if and only if it is composed of two simple chains of clauses like the following ones*:

$$x \vee l_1, \quad \neg l_1 \vee l_2, \quad \ldots, \quad \neg l_m \vee y, \quad \neg y \vee s_1, \quad \ldots, \quad \neg s_m \vee \neg y$$
$$\neg x \vee p_1, \quad \neg p_1 \vee p_2, \quad \ldots, \quad \neg p_m \vee z, \quad \neg z \vee q_1, \quad \ldots, \quad \neg q_m \vee \neg z$$

This lemma shows that every inconsistent and irredundant set of clauses is composed exactly of two chains of clauses, each one not containing the same literal twice. The length of each such chain is at most the number of literals; therefore, the number of clauses of such formula is at most two times the number of literals. Therefore, if an inconsistent 2CNF formula contains a number of clauses that is greater than four times the number of its variables, it is redundant. If it contains less clauses, $\mathrm{O}(nm)$ and $\mathrm{O}(mm)$ are the same. Therefore, checking the consistency of $\Pi \backslash \gamma$ for each $\gamma \in \Pi$ has complexity $\mathrm{O}(nm)$.

## 4.2. Redundancy checking: Consistent 2CNF formulae implying literals

When a formula is consistent but implies some literals, the redundancy of clauses $l \vee l'$ such that $\Pi \models l$ can be checked in linear time: time $\mathrm{O}(m)$ is required for every literal $l$ that is implied by $\Pi$. The redundancy of the other clauses can be then checked by verifying the redundancy of $\Pi \backslash (\Pi|l) \cup \{l\}$ by Lemma 1. Performing this step for every literal that is entailed by $\Pi$ produces the formula $\Pi \backslash (\Pi|\Pi_{\models}) \cup \Pi_{\models}$. The two parts $\Pi \backslash (\Pi|\Pi_{\models})$ and $\Pi_{\models}$ of this formula do not share variables by Lemma 2; the first part does not entail any literal. The redundancy of the first part can therefore be checked using the algorithm of the next section.

Let $l$ be a literal such that $\Pi \models l$. By Lemma 2, we have $\Pi \cup \{\neg l\} \models_{UP} \bot$, which means that unit propagation from $\neg l$ derives both a literal and its negation. In other words, there exists a variable $x$ such that $\Pi \cup \{\neg l\} \models_{UP} x$ and $\Pi \cup \{\neg l\} \models_{UP} \neg x$. By definition, there are then two acyclic paths in the graph of $\Pi$ induced by $\neg l$, one from $\neg l$ to $x$ and one from $\neg l$ to $\neg x$. The first clause of each of these two paths are the only clauses that are necessary to derive $x$ and $\neg x$ from $\neg l$. Regardless of whether these two clauses are the same or not, they are the only clauses containing $l$ that are necessary to prove $\Pi \cup \{\neg l\} \models_{UP} \bot$. As a result, if $l$ is contained in more than two clauses of $\Pi$, this formula is redundant.

In order to check the redundancy of clauses $l \vee l'$ such that $\Pi \models l$, we first check whether the number of such clauses is greater than two. If this is the case, the set is redundant. Otherwise, we check the redundancy of the clauses $l \vee l'$ by simply performing the linear-time entailment check $\Pi \backslash \{l \vee l'\} \models l \vee l'$ for all such clauses $l \vee l'$. Since there are most two such clauses, this test only requires linear time. This test is repeated for all literals $l$ such that $\Pi \models l$; therefore, the total running time is $\mathrm{O}(nm)$.

## 4.3. Redundancy checking: Consistent 2CNF formulae not implying literals

We show that the redundancy of 2CNF consistent formulae not implying literals can be checked in time $\mathrm{O}(nm)$, where $n$ is the number of variables and $m$ is the number of clauses of the formula.

Let $\Pi$ be a consistent 2CNF formula not implying any literal, and let $\neg l \vee l'$ be one of its clauses. Lemma 4 can be simplified thanks to the assumption that no literal is implied: $\Pi \backslash \{\neg l \vee l'\} \models \neg l \vee l'$ holds if and only if $\Pi \backslash \{\neg l \vee l'\} \cup \{l\} \models_{UP} l'$. Indeed, neither $\neg l$ nor $l'$ are implied by $\Pi$, so they cannot be implied by $\Pi \backslash \{\neg l \vee l'\}$ either.

We therefore only have to check whether $\Pi \backslash \{\neg l \vee l'\} \cup \{l\} \models_{UP} l'$, which can be done by checking whether the graph induced by $l$ on $\Pi$ contains a path from $l$ to $l'$ that does not contain the edge corresponding to the clause $\neg l \vee l'$. We now show that this check can be done for all clauses containing $\neg l$ at the same time in $\mathrm{O}(m)$. We recall the definition of $C_\Pi(l)$.

$$C_\Pi(l) = \{l' \mid \neg l \vee l' \in \Pi\}$$

Redundancy of a clause $\neg l \vee l'$ is equivalent to the existence of a path in the graph of $\Pi$ induced by $l$ from another literal in $C_\Pi(l)$ to $l'$. In general, if there is a path not containing $l$ from a node in $C_\Pi(l)$ to another node in $C_\Pi(l)$, the formula is redundant. Consider for example the following graph induced by $l$ on a formula. This formula is redundant, as we can delete the edge $l \rightarrow l''$, and still $l''$ is reachable from $l$, as shown in Fig. 8.

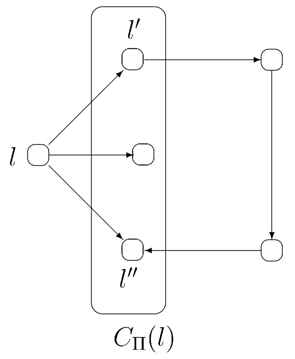

Fig. 8. Clause $\neg l \vee l''$ is redundant.

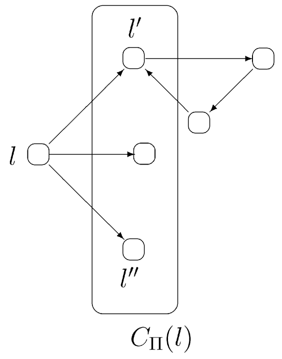

Fig. 9. Reachability from the same node of $C_\Pi(l)$ does not prove redundancy.

The first step of the algorithm is to remove $l$ and all its incident edges from the graph. The second step is that of checking the existence of a path from $C_\Pi(l)$ to $C_\Pi(l)$ in the resulting graph. Note that no pair of opposite literals can be reached from $l$ because $\Pi$ does not entail $\neg l$.

A variant of the algorithm for node reachability can be used for doing this check while visiting the graph only once. The original algorithm for reachability is as follows:

(1) the starting node is marked 0 while all other nodes are marked $\infty$;
(2) at each step, take the set $N_H$ of nodes that are marked with the highest integer; let $i$ be this integer:
   (a) for each $n \in N_H$, consider all its successors $m$;
   (b) mark each $m$ with the minimum among $i + 1$ and its previous marker;
(3) if no label has been changed during Step 2, stop.

The idea is that the label of the node is the distance from the starting node to it. By visiting the graph width-first, we are considering an edge at most once in the whole process. This is why the algorithm is linear.

This algorithm can be applied to the problem of redundancy if the graph is acyclic: start with the nodes in $C_\Pi(l)$ (instead of a single node), and visit the graph until a node in $C_\Pi(l)$ is reached. In other words, if we reach a situation in which the successor $m$ of a node $n \in N_H$ is in $C_\Pi(l)$, then the set of clauses is redundant.

The algorithm does not work for cyclic graphs: the formula is redundant if a node in $C_\Pi(l)$ can be reached from *another* node in $C_\Pi(l)$. On the contrary, if a node in $C_\Pi(l)$ can be reached from itself, the formula is not necessarily redundant: a cycle of this kind does not prove redundancy. Fig. 9 shows a node in $C_\Pi(l)$ reachable from itself; while a node in $C_\Pi(l)$ is reachable from a node in $C_\Pi(l)$, the set is not redundant.

The algorithm must be modified in such a way it checks whether a node of $C_\Pi(l)$ can be reached from *another* node of $C_\Pi(l)$. This can be done by marking each node we visit not only with its distance from $C_\Pi(l)$, but also with the nodes of $C_\Pi(l)$ it is reachable from.

This variant of the node reachability algorithm is however not linear because the set of nodes of $C_\Pi(l)$ a node is reachable from may grow to contain all nodes. However, when a node is reachable from *two* or more different nodes of $C_\Pi(l)$, we do not have to care about the node we started from any longer. Indeed, if a node $l'$ is reachable from two (or more) different nodes $l_1, l_2 \in C_\Pi(l)$, and a node of $l_3 \in C_\Pi(l)$ is reachable from $l'$, then either $l_1$ or $l_2$ is different from $l_3$. As a result, the graph contains a path from $l_1$ to $l_3$ or one from $l_2$ to $l_3$, and at least one between $l_1$ and $l_2$ is different from $l_3$. As a result, once a node $l$ is known to be reachable from two different nodes of $C_\Pi(l)$, we only need to check whether a node of $C_\Pi(l)$ can be reached from $l$.

The algorithm is as follows. In a first phase, labels of nodes can be either $(i, n)$, where $n \in C_\Pi(l)$ and $i$ is an integer, or the special mark two.

(1) each node of $n \in C_\Pi(l)$ is marked with $(0, n)$; take $N_0 = C_\Pi(l)$; set $i = 0$;
(2) set $N_{i+1} = \emptyset$;
(3) let $m \in N_i$, and let $(i, n)$ be its marker; for any of its successors $t$:

   (a) if $t$ is not marked, mark it with $(i+1, n)$, and put $t$ in $N_{i+1}$;
   (b) if $t$ is marked with $(j, n)$, it must be $j \leqslant i$; do not change its marker;
   (c) if $t$ is marked with $(j, s)$, with $n \neq s$, mark it with two;
   (d) if $t$ is marked with two, do not change its mark.

(4) if $N_{i+1}$ is empty, stop; otherwise, set $i = i+1$ and go to step 2.

This is almost the usual visit of the graph width-first. The point is that we mark the nodes not only with their distance from $C_\Pi(l)$, but also with the node they can be reached from. Whenever a node is found out to be reachable from two nodes of $C_\Pi(l)$, we mark it with two and do not continue the search from it. If a node $m \in C_\Pi(l)$ is the successor of a node marked with $(n, i)$, and $n \neq m$, then the graph is redundant. Note that the whole algorithm is linear, as each edge is at most traversed once.

We now have to visit the successors of the nodes we have marked with two, and check whether any node of $C_\Pi(l)$ can be reached from them. This can be done with the very same original reachability algorithm, which is still linear in time.

This algorithm determines the redundancy of all clauses containing one literal in time $O(m)$. Therefore, all clauses can be checked in time $O(nm)$.

## 5. Irredundant equivalent subsets (I.E.S.'s)

The following problems about I.E.S.'s are polynomial for 2CNF formulae because of the polynomiality of entailment for this restriction.

*check whether a formula is an* I.E.S. *of another formula.* Check containment and irredundancy;
*check whether a clause is in all* I.E.S.*'s.* A clause $\gamma$ is in all I.E.S.'s of a CNF formula $\Pi$ if and only if $\Pi \backslash \{\gamma\} \not\models \gamma$ [17];
*uniqueness.* A CNF formula $\Pi$ has an unique I.E.S. if and only if $\{\gamma \in \Pi \mid \Pi \backslash \{\gamma\} \not\models \gamma\} \models \Pi$ [17].

Two other problems require a more detailed analysis: the size of I.E.S. (checking whether a formula has an I.E.S. of size bounded by a number $k$) and the presence of a clause in an I.E.S. (checking whether a given clause is present in an I.E.S. of a given formula).

As it is clear from the two tables below, complexity, in the cases that have been successfully classified, does not depend on whether the formula is consistent or implies literals. However, the proofs are different in the various cases. In the two tables below, "single" indicates formulae implying a single literal and "non-single" indicates formulae not implying any literal.

| | Size of I.E.S. | | |
|---|---|---|---|
| | inconsistent | single | non-single |
| acyclic | P | P | P |
| cyclic | ?? | NP-hard | NP-hard |

| | Presence in a I.E.S. | | |
|---|---|---|---|
| | inconsistent | single | non-single |
| acyclic | P | P | P |
| cyclic | NP-hard | NP-hard | NP-hard |

We use the following order of the cases: first, all acyclic cases, then all the cyclic cases; in each case, we first consider inconsistent formulae, then consistent formulae implying literals, and then consistent formulae not implying literals. In the rest of this section, we prove the results that easily follow from previous ones.

The problems about inconsistent and acyclic 2CNF formulae can be easily proved polynomial. Acyclicity implies that the set of binary clauses is consistent by Theorem 1. Therefore, Lemma 8 and Lemma 9 apply: every I.E.S.

contains at most two unary clauses, and every I.E.S. can be proved inconsistent by applying unit propagation from its unary clauses.

This leads to the following algorithm for building an I.E.S. of minimal size: for every pair of literals, calculate the minimal number of binary clauses (if any) that are necessary to derive the second literal from the first using unit propagation. This can be done using an algorithm similar to the algorithm for the minimal distance in a graph. Given these distances, one can compute, for every unit clause $l$ and literals $l'$, $\neg l'$, the minimal number of clauses needed to derive $l'$ from $l$ and $\neg l'$ from $l'$. These two chains cannot contain the same clause by Lemma 5, so this is the minimal number of clauses needed to derive inconsistency by unit propagation from $l$. In a similar way, one can compute the minimal number of clauses to reach the negation of a unit clause from another unit clause.

As for the presence of a clause in an I.E.S., one can proceed as follows: for every unit clause and pair of opposite literals, check whether there exists a path that comprises the clause and goes from the unit clause to the pair of opposite literals. In the same way, one checks whether there exists a path from a unit clause to the negation of another unit clause.

The following two results about consistent acyclic 2CNF formulae have already been proved.

- Acyclic consistent formulae not implying single literals have a unique I.E.S.
- Building an I.E.S. for an acyclic consistent formula implying the literals $l_1, l_2, \ldots, l_n$ can be done by choosing independently some of the clauses containing $l_1$, some of the clauses containing $l_2$, etc.

These two facts allow proving that, for acyclic consistent 2CNF formulae, one can determine presence and size of I.E.S.'s in polynomial time.

**Theorem 3.** *The unique* I.E.S. *of a consistent acyclic 2CNF formula not implying literals can be found in quadratic time.*

**Proof.** Let $\Pi$ be a consistent acyclic 2CNF formula. Since it has a single I.E.S., a clause is in its I.E.S. if and only if it is in all its I.E.S.'s. Since checking presence in all I.E.S.'s can be done by checking $\Pi\backslash\{\gamma\} \models \gamma$, and this inference is linear-time for 2CNF formulae, we can conclude that checking the presence of each clause in the unique I.E.S. of the formula can be done in linear time. □

The proof of the theorem shows a quadratic algorithm for generating the single I.E.S. of $\Pi$ based only one the uniqueness of such an I.E.S. However, Lemma 4 allows for a slightly better algorithm: for each $l$, we can check all clauses $\neg l \vee l_1$ at once by visiting the graph $R_\Pi(l)$. Since the construction and visit of $R_\Pi(l)$ takes linear time, the overall running time is $O(nm)$, where $n$ is the number of variables and $m$ is the number of clauses of the formula.

**Theorem 4.** *The problems of presence of a clause in an* I.E.S. *and of existence of an* I.E.S. *of given size are polynomial-time for acyclic consistent 2CNF formulae.*

**Proof.** By Lemma 18, the presence of a clause $l \vee l'$ with $\Pi \models \neg l$ in an I.E.S. only depends on whether $l' \in S_\Pi(l)$ or there exists $l''$ such that $(l', l'') \in P_\Pi(l)$. Since $S_\Pi(l)$ and $P_\Pi(l)$ can be checked in polynomial time by definition, checking the presence of a clause $l \vee l'$ in an I.E.S. is a polynomial-time problem if $\Pi \models \neg l$. All other clauses can be checked in polynomial time thanks to Corollary 3 and Theorem 3.

One of the smallest I.E.S.'s of an acyclic formula can be built in a similar way but choosing $\neg l$ or a clause $\neg l \vee l'$ such that $l' \in S_\Pi(l)$ if possible, and a pair of clauses $\neg l \vee l'$ and $\neg l \vee l''$ with $(l', l'') \in P_\Pi(l)$ otherwise. To account for the pair of clauses $\neg l \vee l'$ and $\neg l \vee \neg l'$ that replace the unit clause $l$ by the transformation of Section 3.2, we count such a pair as it were a single clause. □

## 6. Size of I.E.S.

In this section, we consider the problem of checking whether a formula has an I.E.S. of size bounded by a given integer $k$. This problem has already been proved polynomial for inconsistent and consistent acyclic 2CNF formulae. In this section, we prove the results for consistent cyclic formulae; the case for inconsistent cyclic formula is open.

## 6.1. Size of I.E.S.: Cyclic consistent 2CNF formulae implying literals

We show that Lemma 18 does not hold if the literal $l$ under consideration is in a cycle of clauses. Consider the following formula:

$$\Pi = \{\neg l' \vee l, \neg l \vee l', \neg l \vee l'', \neg l' \vee x, \neg l' \vee \neg x, \neg l'' \vee y, \neg l'' \vee \neg y\}$$

The graph of $\Pi$ induced by $l$ is shown in Fig. 10, which clearly shows a simple cycle between the literals $l$ and $l'$.

For this formula, $S_\Pi(l) = \{l', l''\}$, while $P_\Pi(l) = \emptyset$. According to Lemma 18, if $\Pi'$ is an I.E.S. of $\Pi$, then we can replace $\Pi' \cap D_\Pi(l)$ with a single clause containing a literal of $S_\Pi(l)$, and what results is still an I.E.S. of $\Pi$. We show a counterexample. The formula $\Pi$ is equivalent to $\{\neg l, \neg l', \neg l''\}$. The following is an I.E.S. of $\Pi$.

$$\Pi' = \{\neg l' \vee l, \neg l \vee l'', \neg l'' \vee y, \neg l'' \vee \neg y\}$$

$\Pi'$ is represented by the graph in Fig. 11. This subset is equivalent to $\Pi$ because it entails all three literals $\neg l$, $\neg l'$, and $\neg l''$. Indeed, $\neg l''$ is entailed by the clauses $\neg l'' \vee y$ and $\neg l'' \vee \neg y$, while $\neg l$ and $\neg l'$ are entailed by $\neg l' \vee l$ and $\neg l \vee l''$, which make $l''$ reachable by unit propagation from $l$ and $l'$.

According to Lemma 18, we should be able to obtain another I.E.S. from $\Pi'$ by replacing all clauses of $D_\Pi(l)$ with $\neg l \vee l'$ in it, as $l' \in S_\Pi(l)$. Let $\Pi'' = \Pi' \backslash D_\Pi(l) = \{\neg l' \vee l, \neg l'' \vee y, \neg l'' \vee \neg y\}$. If Lemma 18 were true for cyclic formulae, it would hold that $\Pi'' \cup \{\neg l \vee l'\}$ is another I.E.S. of $\Pi$. This is however false, as this set does not imply neither $\neg l$ nor $\neg l'$. Graphically, $\Pi'' \cup \{\neg l \vee l'\}$ is represented in Fig. 12.

This formula still entails $\neg l''$, but no longer entails either $\neg l$ or $\neg l'$. These are indeed the two literals involved in the only cycle of this formula. Intuitively, Lemma 18 does not hold for cyclic formulae because, while $\Pi' \cup \{l'\} \models_{UP} \perp$ still holds in any I.E.S. $\Pi'$ of $\Pi$ for every $l' \in S_\Pi(l)$, the unit derivation of $\perp$ from $l'$ might involve the literal $l$. Therefore, the choice of clauses containing $l$ cannot be done independently on the choices of the other clauses.

We show that the problem of checking the existence of an I.E.S. of a formula is NP-complete and remains NP-hard even if the formula entails a literal for all variables but one. The following lemma is a needed preliminary result that is obtained by applying a result by Fortune et al. [8].

**Lemma 20.** *Deciding whether a graph contains a simple cycle including two given nodes is* NP-*complete.*

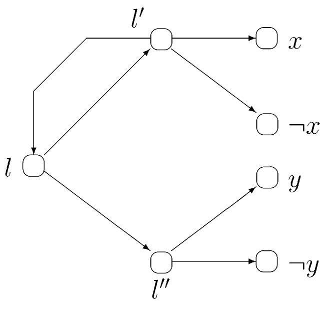

Fig. 10. The graph of $\Pi$ induced by $l$.

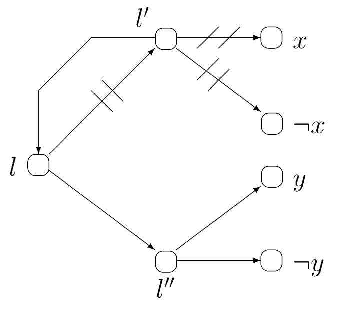

Fig. 11. An I.E.S. of the formula of Fig. 10.

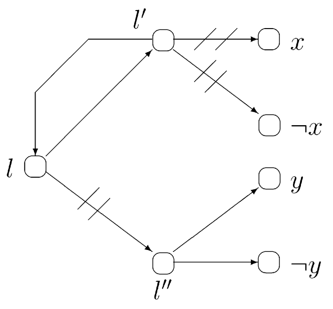

Fig. 12. The graph of $\Pi''$ induced by $l$ shows that $\Pi''$ is not an I.E.S. of $\Pi$.

**Proof.** Membership is obvious: guess a set of edges and check whether it is a simple cycle including the two nodes. Hardness holds because the presence of such cycle is equivalent to the homomorphism of the graph $G = \langle\{a, b\}, \{a \rightarrow b, b \rightarrow a\}\rangle$ with a subset of the graph. This problem is NP-hard because $G$ does not contain a node that is the head of all edges or a node that is the tail of all edges [8]. □

This theorem shows that the problem of checking whether a set of clauses implying a single literal, and containing cycles in the graph induced by the negation of this literal, is NP-complete.

**Theorem 5.** *Deciding whether a consistent 2CNF formula has an* I.E.S. *of size $k$ is* NP*-complete and remains* NP*-hard even if the formula implies a literal for every variable but one.*

**Proof.** Hardness is proved by reduction from the problem of checking the existence of a simple cycle in a graph containing two given nodes $x$ and $y$. We assume that $y$ is reachable from $x$ and vice versa, and that all nodes can be reached and reach both $x$ and $y$. If these conditions are not met, either the problem becomes trivial or the graph can be purged of some nodes.

The reduction is as follows: for each node of the graph there is a variable. Another variable $z$ is also used. For each edge $a \rightarrow b$ of the graph we have a clause $\neg a \vee b$. We also add the two clauses $\neg x \vee z$ and $\neg y \vee \neg z$, where $x$ and $y$ are the nodes that are checked for being in the same simple cycle. This set of clauses satisfy the condition of the theorem, as both $z$ and $\neg z$ are reachable from any other positive literal.

Let $k$ be the number of nodes of the graph plus two. Let us assume that the graph contains a simple cycle including both $x$ and $y$. In this case, an I.E.S. of size $k$ is composed of the following clauses: $\neg x \vee z$ and $\neg y \vee \neg z$; all clauses corresponding to the edges of the cycle; and a clause for every other node obtained by recursively adding a clause connecting a node to the cycle if not already. This I.E.S. is of size $k$ because it contains a clause associated to an outgoing edge of a node for each node of the original graph, with the only exception of $x$ and $y$, for which the I.E.S. also contains the clauses $\neg x \vee z$ and $\neg y \vee \neg z$.

Let us assume that the graph does not contain simple cycles including both $x$ and $y$. Since removing either $\neg x \vee z$ or $\neg y \vee \neg z$ makes the formula not to imply literals any longer, any I.E.S. of this formula contains these two clauses; more precisely, both $z$ and $\neg z$ must be reachable from any other positive literal in every I.E.S. of the formula. As a result, $x$ must be reachable from $y$ and vice versa. However, since the graph does not contain any simple cycle, at least one node $w$ has two outgoing edges in these two paths. Since every positive literal but $z$ must be associated an outgoing edge in every I.E.S. of the formula, the total number of edges needed to form an I.E.S. is at least $k + 1$. □

## *6.2. Size of* I.E.S.*: Cyclic consistent formulae not implying literals*

The problem of determining whether a formula has an I.E.S. of size at most $k$ is NP-complete if the formula is consistent and cyclic. This is in particular true even if the formula does not entail any literal.

**Theorem 6.** *Deciding whether a 2CNF formula has an* I.E.S. *of size bounded by $k$ is* NP*-complete, and remains* NP*-hard even if the formula implies that all variables are equivalent and does not imply any single literal.*

**Proof.** The problem of finding a minimum equivalent subgraph is NP-complete. In particular, it remains complete even if the graph is strongly connected [14,24]. This is the problem of finding the minimum number of edges of the graph that makes the resulting graph strongly connected.

We show a reduction from this problem to that of checking whether there exists an I.E.S. of a 2CNF formula $\Pi$ of size bounded by $k$. Let $G = (N, E)$ be a graph. The set of variables of $\Pi$ is the set of nodes $N$. For each edge $(i, j)$ the set $\Pi$ contains the clause $\neg i \vee j$. For any two nodes $i$ and $j$, the node $j$ is reachable from $i$ if and only if $\Pi \cup \{i\} \models_{UP} j$. In order to complete the proof, we only have to show that $\Pi \not\models l$ for any literal $l$: if this is true, then reachability is in one-to-one correspondence with entailment, thus showing that equivalence of graphs implies equivalence of the corresponding formulae and vice versa.

Since each clause contains a positive and a negative literal, it is satisfied both by the model setting all variables to true and by the model setting all variables to false. These are therefore both models of $\Pi$. However, if one of them satisfies a literal $l$, the other one does not. Therefore, no literal $l$ is implied by $\Pi$. □

## 7. Presence in an I.E.S.

In this section, we study the problem of checking whether a clause is contained in some I.E.S.'s of a given 2CNF formula. For acyclic sets, the problem has already been proved polynomial. This result holds even if some cycles are present but none include the clause to check, in the case of consistent formulae; a formal proof of this statement is given in Appendix A.8. We now prove that the problem is NP-complete for cyclic formulae, regardless of whether the formula is consistent or implies some literals.

### *7.1. Presence in an* I.E.S.*: Cyclic inconsistent 2CNF formulae*

The problem of deciding the presence of a clause in an I.E.S. of an inconsistent 2CNF formula is NP-hard if the formula contains cycles. We use this simple preliminary lemma.

**Lemma 21.** *The problem of checking in a graph the existence of a simple path from node $x$ to node $y$ including a given edge is* NP-*complete.*

We can now prove that the problem of presence of a clause in an I.E.S. is NP-complete if the formula is inconsistent and cyclic.

**Theorem 7.** *Deciding whether a cyclic inconsistent 2CNF formula has an* I.E.S. *that contains a given clause is* NP-*complete.*

**Proof.** Membership is obvious, as the problem can be solved by guessing a subset of the formula and then checking whether it is an I.E.S. and contains the given clause. Hardness is proved by reduction from the problem of checking whether a graph contains a simple path from a node to another one that includes a given edge.

Given the instance of the original problem $(G, x, y, a, b)$, in which we want to determine whether there exists a simple path from $x$ to $y$ including the edge $a \to b$, we build a formula $\Pi$ as follows: for each node of the graph, we have a variable; for each edge $n \to m$ of the graph, we have the clause $\neg n \vee m$; finally, we add the following clauses, where $z$ and $w$ are new variables:

$$\{\neg y \vee z, \neg y \vee \neg z, x \vee w, x \vee \neg w\}$$

Graphically, $\Pi$ can be represented as in Fig. 13.

Removing even a single clause containing $z$ or $w$ makes the formula satisfiable. Therefore, these clauses are in all I.E.S.'s of the formula. Since a 2CNF formula is unsatisfiable if and only if $\Pi \cup \{w\} \models_{UP} \bot$ and $\Pi \cup \{\neg w\} \models_{UP} \bot$ for some variable $w$, these four clauses must necessarily be all included in the paths from $w$ and $\neg w$ to $\bot$, as otherwise one of these clauses would be redundant. As a result, $\Pi'$ is an I.E.S. of $\Pi$ if and only there exists a literal $l$ such that $y$ is reachable from $l$ and $\neg x$ is reachable from $\neg l$. The second condition is equivalent to $l$ being reachable from $x$; therefore, the two conditions are equivalent to the existence of a path from $x$ to $y$. As a result, $\Pi'$ is an I.E.S. of $\Pi$ if and only if it contains the four clauses containing $z$ and $w$, and a path from $x$ to $y$.

If this path from $x$ to $y$ in a subset of the formula is cyclic, the formula is not irredundant because all clauses in the cycle can be removed while preserving the reachability of $y$ from $x$ and therefore preserving unsatisfiability. As a result, the I.E.S.'s of the formula are in a bijection with the simple paths from $x$ to $y$. Such a formula contains $\neg a \vee b$ if and only if there exists a simple path from $x$ to $y$ including the edge $a \to b$. □

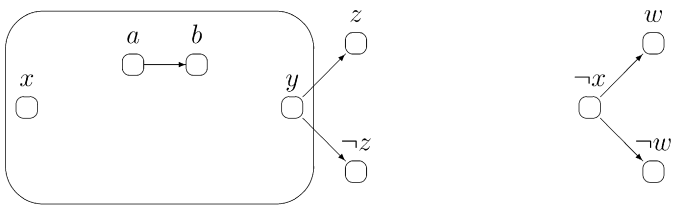

Fig. 13. Graphical representation of $\Pi$.

## *7.2. Presence in an* I.E.S.*: Cyclic consistent 2CNF formulae implying literals*

In this section we consider the problem of presence of a clause in an I.E.S. of a cyclic consistent 2CNF formula implying some literals. The case in which no cycle includes the clause to check is proved polynomial in Appendix A.8. We now show that the general problem is NP-complete.

**Theorem 8.** *The problem of checking the presence of a clause in some* I.E.S.*'s of a consistent 2CNF formula $\Pi$ is* NP*-complete and remains hard even if $\Pi \models l$ for every positive literal $l$ but one.*

**Proof.** The proof is by reduction from the problem of deciding the existence of two vertex-disjoint paths in a directed graph, which is NP-complete [6,8] (the corresponding problem for undirected graphs is polynomial [23].) This is the problem of establishing whether a graph $G$ contains a path from node $s_1$ to node $t_1$ and a path from node $s_2$ to node $t_2$ and these two paths do not share nodes. We can assume that, from each node of the graph, either $t_1$ or $t_2$ is reachable; otherwise, the nodes not satisfying this condition can be removed from the graph in polynomial time. The formula we consider is the one corresponding to the graph in Fig. 14.

This formula implies the negation of all positive literals beside $x$ because the pair of nodes $x$ and $\neg x$ can be reached from any other node of the graph. The same property must therefore be true for all I.E.S.'s of this formula. Since $x$ and $\neg x$ form the only pair of opposite literals in the graph, the property holds only if both $l_3$ and $l_4$ are reachable from any other node besides $x$ and $\neg x$ in the graph corresponding to the I.E.S.

If $G$ has a pair of vertex-disjoint paths from $s_1$ to $t_1$ and from $s_2$ to $t_2$, respectively, there exists an I.E.S. containing the clause $\neg l_1 \vee l_2$. Let us consider the subformula $\Pi'$ built as follows. First, $\Pi'$ contains all clauses corresponding to the edges of the subgraph in Fig. 15.

Second, $\Pi'$ contains a number of clauses corresponding to edges making either $t_1$ or $t_2$ reachable from every node of $G$ not having already this property: this can be done by adding a clause $\neg s \vee t$ if either $t_1$ or $t_2$ is reachable from $t$

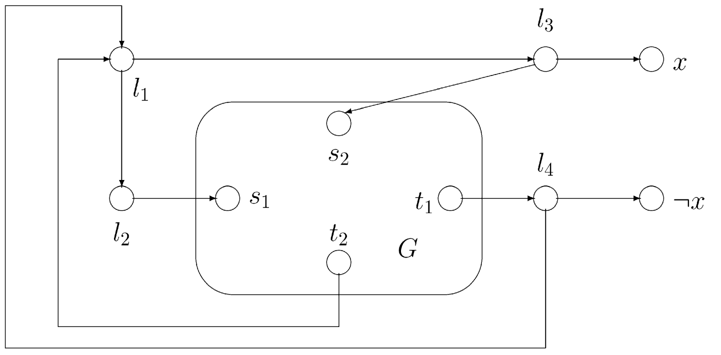

Fig. 14. The graph of the formula used in the proof.

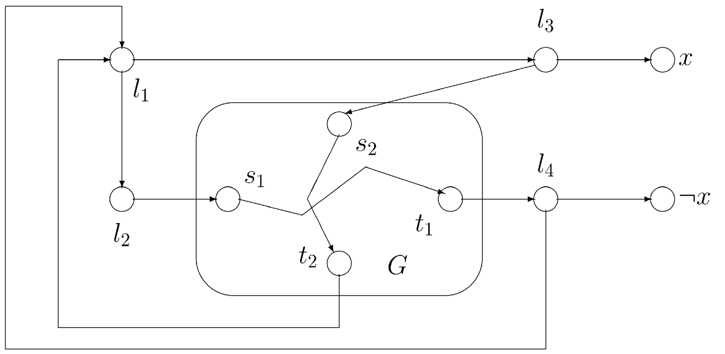

Fig. 15. The first part of $\Pi'$.

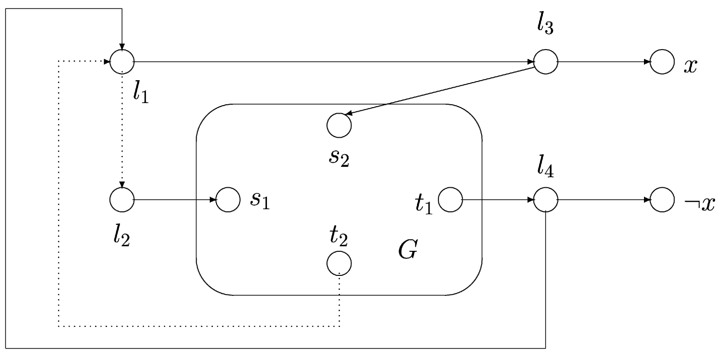

Fig. 16. The edges that are necessarily contained in any I.E.S. of $\Pi$.

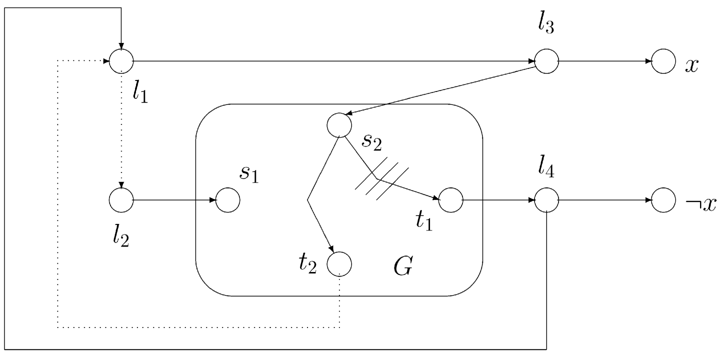

Fig. 17. $s_2$ is connected to $t_2$ but not to $t_1$.

but not from $s$. The addition of these clauses does not make $t_2$ reachable from $s_1$ or $t_1$ from $s_2$ because a clause $\neg s \vee t$ is only added if no other edge outgoing from $s$ is already in the I.E.S.

The subformula $\Pi'$ makes both $l_3$ and $l_4$ reachable from $l_1$ and $l_1$ reachable from any other node of the graph. As a result, $\Pi' \equiv \Pi$. Since $l_4$ is only reachable from $l_1$ via the edge from $l_1$ to $l_2$, the corresponding clause is irredundant in $\Pi'$. Since $\neg l_1 \vee l_2$ is irredundant in $\Pi'$, every I.E.S. of $\Pi'$ contains this clause by Property 1. Since $\Pi' \equiv \Pi$, every I.E.S. of $\Pi'$ is also an I.E.S. of $\Pi$. Therefore, $\neg l_1 \vee l_2$ is contained in some I.E.S.'s of the original formula.

Let us now assume that $G$ contains no vertex-disjoint paths from $s_1$ to $t_1$ and from $s_2$ to $t_2$, respectively. We prove that $\neg l_1 \vee l_2$ is redundant in every I.E.S. of the formula. Since $x$ is the only variable that occurs both direct and negated in the graph, in every I.E.S. of $\Pi$ both $l_3$ and $l_4$ are reachable from any other variable besides $x$ and $\neg x$. Fig. 16 shows the edges that are necessarily contained in any I.E.S. of $\Pi$ because either they are the only outgoing edges of a node or they are the only edges that are incoming to $l_3$ or $l_4$.

Since $s_2$ must be connected to nodes outside $G$ (i.e., $l_3$ and $l_4$), it is either connected to $t_1$ or to $t_2$. If $t_1$ is reachable from $s_2$, then both $l_3$ and $l_4$ are reachable from $l_1$ without using the clause $\neg l_1 \vee l_2$, which is therefore redundant. We can therefore assume that $s_2$ is connected to $t_2$ but not to $t_1$, as shown in Fig. 17.

By assumption, $t_1$ is not reachable from $s_2$ using only edges inside $G$. The node $s_1$ can therefore be connected to $l_4$ only via a path starting from $t_2$. However, such a path can only go to $l_1$, then $l_2$, and finally $s_1$. By assumption, however, every path from $s_1$ to $t_1$ shares a node with the path from $s_2$ to $t_2$, which makes $t_1$ reachable from $s_2$, contradicting the assumption. □

## *7.3. Presence in an* I.E.S.*: Cyclic consistent 2CNF formulae not implying literals*

The problem of checking the presence of a clause in an I.E.S. of a 2CNF formula is polynomial if the formula is acyclic and consistent, and does not entail any single literal. We now show that the same problem is NP-complete if the formula is still consistent but cyclic. First of all, if a formula is consistent and does not entail literals, then every clause $l_1 \vee l_2$ can be made irredundant by the following lemma.

**Lemma 22.** *If $\Pi$ does not entail $\neg l_1$ nor $\neg l_2$ and contains the clauses $l_1 \vee w$ and $\neg w \vee l_2$, where $w$ is a variable appearing only in these two clauses, then $\Pi$ is equivalent to $\Pi \cup \{l_1 \vee l_2\}$, and the two clauses $l_1 \vee w$ and $\neg w \vee l_2$ are irredundant in $\Pi$.*

**Proof.** Equivalence is due to the fact that $l_1 \vee l_2$ is obtained from $l_1 \vee w$ and $\neg w \vee l_2$ by resolution. Irredundancy is due to the fact that $\Pi \cup \{\neg l_1\} \models_{UP} w$ in $\Pi$ because $\Pi$ is consistent and does not entail literals. Since $l_1 \vee w$ is the only clause containing $w$ positively, this clause is necessary. For the same reason, $\neg w \vee l_2$ is irredundant. □

If a clause is necessary in a formula, it is contained in all its I.E.S.'s. Whenever we have a formula $\Pi$ that is consistent and does not entail literals, and we want a clause $l_1 \vee l_2$ to be contained in all its I.E.S.'s, we can replace it with $l_1 \vee w$ and $\neg w \vee l_2$. The original clause is still implied by these two new ones by resolution, but the new clauses are both irredundant. In order to keep proofs simple, we use the following notation.

**Notation.** $l_1 \overset{\circ}{\vee} l_2$ means $\{l_1 \vee w, \neg w \vee l_2\}$ where $w$ is a new variable.

We use this notation because the clauses $\{l_1 \vee w, \neg w \vee l_2\}$ actually represent the single clause $l_1 \vee l_2$ since $w$ is not used anywhere else. The circle over the symbol $\vee$ reminds us that these two clauses cannot be removed from a formula without changing its semantics. We can now prove that deciding the presence of a clause in an I.E.S. of a formula is NP-complete.

**Theorem 9.** *Deciding whether a clause is in an* I.E.S. *of a cyclic and consistent 2CNF formula $\Pi$ is* NP*-hard even if no single literal is implied by $\Pi$ and $\Pi$ makes all literals equivalent.*

**Proof.** We show a proof of hardness from 3sat. The set of clauses generated by this particular reduction is such that all clauses are of the form $\neg l \vee l'$. Such formulae can be represented by their induced graphs. In this proof, we use $l \Rightarrow l'$ to denote the reachability of $l'$ from $l$ and $l \Leftrightarrow l'$ to denote mutual reachability ($l$ and $l'$ can be reached from each other).

Given a set of clauses $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$, we generate a formula $\Pi$ and one of its clauses $\neg l_1 \vee l_2$ in such a way $\neg l_1 \vee l_2$ is in some I.E.S.'s of $\Pi$ if and only if $\Gamma$ is satisfiable. The graph corresponding to $\Pi$ is strongly connected. In particular, truth assignments on $\Gamma$ correspond to subsets of $\Pi$ in which $l_2 \Rightarrow n \Rightarrow l_1$ for every node $n$. Therefore, the graph is strongly connected if and only if $l_1 \Rightarrow l_2$. The truth assignment satisfies $\Gamma$ if and only if $l_1 \Rightarrow l_2$ does not hold in the corresponding subformula, thus making the clause $\neg l_1 \vee l_2$ necessary.

From now on, we consider $\Pi$ as a graph. The graph corresponding to $\Gamma$ is as follows: for each variable $x_i$, we have three nodes $x_i$, $x_i^+$, and $x_i^-$. For each clause $\gamma_j$ we have a node $c_j$. The clauses of $\Pi$ are the following ones:

(1) The clause to be checked for presence in an I.E.S.: $\neg l_1 \vee l_2$;
(2) Nodes forming a strongly connected component containing $l_1$:
  (a) $\neg l_1 \overset{\circ}{\vee} x_i$;
  (b) $\neg x_i \vee x_i^+$ and $\neg x_i \vee x_i^-$;
  (c) $\neg x_i^+ \overset{\circ}{\vee} l_1$ and $x_i^- \overset{\circ}{\vee} l_1$;
(3) Nodes forming a strongly connected component containing $l_2$:
  (a) $\neg l_2 \overset{\circ}{\vee} x_i^+$ and $\neg l_2 \overset{\circ}{\vee} x_i^-$;
  (b) $\neg x_i^+ \vee c_j$ if $x_i \in \gamma_j$;
  (c) $\neg x_i^- \vee c_j$ if $\neg x_i \in \gamma_j$.
  (d) $\neg \gamma_j \overset{\circ}{\vee} l_2$.

This graph is strongly connected. This must also be true for every graph representing an I.E.S. of $\Pi$. As an example, the clause $\gamma_1 = x_1 \vee \neg x_2$ is represented as in Fig. 18 (this example uses a binary clause; clauses of three literals are converted in a similar way). The nodes $l_1$ and $l_2$ have been omitted to keep the figure simple: if a node is missing at the left of an arrow, it is $l_1$; if it is at the right, it is $l_2$. Arrows marked with a circle cannot be removed because they correspond to irredundant clauses.

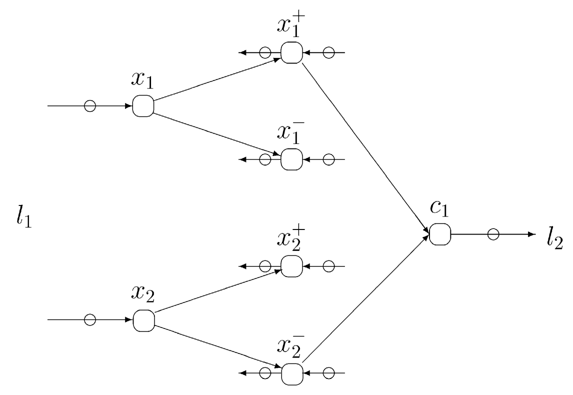

Fig. 18. The subgraph corresponding to $\gamma_1 = x_1 \vee \neg x_2$. A binary clause has been used to make the figure simpler, but clauses of three literals can be converted in a similar way.

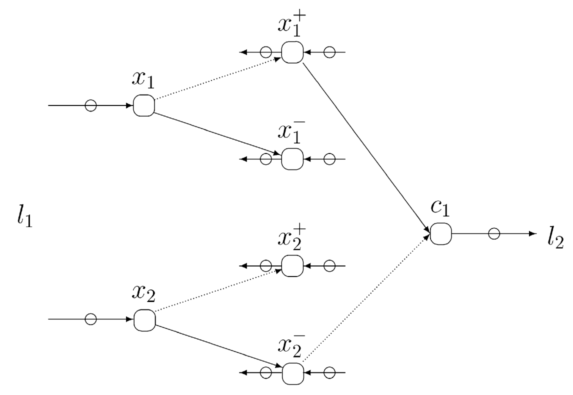

Fig. 19. Dotted lines are clauses removed if $M$ assigns true to both $x_1$ and $x_2$.

Every truth assignment on $\Gamma$ can be associated to a set of clauses to remove from $\Pi$ to the aim of obtaining an I.E.S. In particular, if $\Pi$ is satisfiable we can remove some of the clauses in such a way that $l_2 \Rightarrow n \Rightarrow l_1$ still holds for every node $n$, but $l_1 \Rightarrow l_2$ does not. This way, $\neg l_1 \vee l_2$ is necessary to make this graph strongly connected and therefore equivalent to the original one.

Let us assume that $\Gamma$ is satisfiable, and let $M$ be one of its models. We build a subset of $\Pi$ by removing the following clauses:

(1) if $x_i$ is positive in $M$, remove the clause $\neg x_i \vee x_i^+$ and all clauses from $x_i^-$ to a node $c_j$;
(2) if $x_i$ is negative in $M$, do the other way around.

Fig. 19 shows the clauses that are removed from the formula above if $M$ assigns true to both $x_1$ and $x_2$. In this subgraph, $l_2 \Rightarrow n \Rightarrow l_1$ holds for all nodes. For nodes $x_i^+$ and $x_i^-$ there are edges from $l_2$ to them and from them to $l_1$. The cycle $l_1 \Rightarrow x_i \Rightarrow x_i^+ \Rightarrow l_1$ or the cycle with $x_i^-$ replacing $x_i^+$ causes all nodes $x_i$ to be in the same strongly connected component as $l_1$ if $x_i$ is negative in $M$. A node $c_j$ is in the same strongly connected component as $l_2$ thanks to a cycle $\neg l_2 \vee x_i^+, \neg x_i^+ \vee c_j, \neg c_j \vee l_2$ if $x_i$ is positive in $M$ and in $c_j$, or a similar cycle if the literal of $c_j$ that is true in $M$ is a negative one.

Since $l_2 \Rightarrow n \Rightarrow l_1$ holds for all nodes, the addition of the clause $\neg l_1 \vee l_2$ makes all nodes reachable from each other, making this formula equivalent to the original one. On the other hand, $l_1 \Rightarrow l_2$ is not true in this graph: indeed, all paths $l_1 \vee x_i, \neg x_i \vee x_i^+, \neg x_i^+ \vee c_j, \neg c_j \vee l_2$ have been broken because either the clause $\neg x_i \vee x_i^+$ or the clause $\neg x_i^+ \vee c_j$ has been removed, and the same holds for the similar paths containing $x_i^-$ in place of $x_i^+$.

Let us now assume that $\Pi$ has an I.E.S. $\Pi'$ containing the clause $\neg l_1 \vee l_2$, and show a truth assignment satisfying all clauses of $\Gamma$. Since the graph of $\Pi$ is strongly connected, the same holds for $\Pi'$. Since removing $\neg l_1 \vee l_2$ makes the graph of $\Pi'$ not strongly connected, we have that:

(1) $l_2$ is not reachable from $l_1$ in the graph corresponding to $\Pi' \backslash \{\neg l_1 \vee l_2\}$;
(2) every node is reachable from either $l_1$ or $l_2$ and reaches either $l_1$ or $l_2$ in the graph corresponding to $\Pi' \backslash \{\neg l_1 \vee l_2\}$: otherwise, the addition of $\neg l_1 \vee l_2$ would not make the graph strongly connected.

Since $x_i$ is reachable from $l_1$, it must reach either $l_1$ or $l_2$; however, it cannot reach $l_2$ as otherwise we would have $l_1 \Rightarrow l_2$. As a result, for every $x_i$, either $\neg x_i \vee x_i^+$ or $\neg x_i \vee x_i^-$ is in $\Pi'$. In other words, it cannot be that both clauses are not in $\Pi'$ for the same $x_i$. As a result, the following is a partial model (a consistent set of literals).

$$M = \{x_i \mid \neg x_i \vee x_i^- \notin \Pi'\} \cup \{\neg x_i \mid \neg x_i \vee x_i^+ \notin \Pi'\}$$

We show that all clauses of $\Gamma$ are satisfied by $M$. Let $\gamma_j$ be a clause. Since the clause $\neg c_j \vee l_2$ is not removable, $l_1 \Rightarrow c_j$ cannot hold, as otherwise $l_2$ would be reachable from $l_1$. As a result, $l_2 \Rightarrow c_j$ must hold. This implies that there exists an index $i$ such that either $x_i \in \gamma_j$ and $\neg x_i^+ \vee c_j \in \Pi'$ or $\neg x_i \in \gamma_j$ and $\neg x_i^- \vee c_j \in \Pi'$. In the first case, $\neg x_i \vee x_i^+ \notin \Pi'$ as this clause would create a path from $l_1$ to $l_2$. This however implies that $x_i$ is set to true by $M$. Since $x_i \in \gamma_j$, the clause $\gamma_j$ is satisfied by $M$. The case $\neg x_i \in \gamma_j$ and $\neg x_i^- \vee c_j \in \Pi'$ is similar. □

## 8. Horn formulae

When considering the complexity of redundancy for Horn formulae, the following problems are clearly polynomial:

(1) checking redundancy;
(2) a set is an I.E.S.;
(3) a clause is in all I.E.S.'s;
(4) uniqueness.

The problem of size is easily proved to be NP-complete: indeed, the corresponding proof for the case of consistent acyclic 2CNF formulae not implying single literals uses clauses corresponding to the edges of a graph whose nodes are all positive literals. These clauses are therefore all in the form $\neg x \vee y$, that is, they are binary Horn clauses.

We show that the problem of size is NP-complete for inconsistent Horn formulae.

**Theorem 10.** *Deciding whether a Horn formula $\Pi$ has an equivalent subset of size $k$ is* NP*-complete.*

**Proof.** Membership is obvious. Hardness is proved by reduction from vertex cover. Let $G$ be a graph. We build the following set of Horn clauses: for each node $i$ we have a unit clause $x_i$; for each edge $z = (i, j)$ we have two clauses $\neg x_i \vee a_z$ and $\neg x_j \vee a_z$; finally, we have the clause $\neg a_1 \vee \cdots \vee \neg a_m$.

Inconsistent subsets of this formula are composed of $\neg a_1 \vee \cdots \vee \neg a_m$, plus a pair $x_i$ and $\neg x_i \vee a_z$ for each edge $z$. This means that we have exactly one clause $\neg x_i \vee a_z$ for each edge of the graph. Moreover, for each edge we have to include a unit clause corresponding to one of its incident nodes. Therefore, minimal inconsistent subsets are in one-to-one correspondence with vertex covers of the original graph. Namely, $G$ has a vertex cover of $k$ nodes if and only if the formula has an inconsistent subset of $m + 1 + k$ clauses, where $m$ is the number of edges of $G$. □

## 9. Conclusions

The analysis of the problems of redundancy presented in this paper shows an interesting pattern: the complexity of some problems is related to their *cyclicity*: some problems are polynomial on acyclic problems and NP-complete on cyclic ones. Interestingly, complexity does not instead depends on whether the considered formula is consistent.

Quantitatively, we have shown that checking redundancy can be done with an algorithm that is slightly more efficient than the trivial one. In particular, while checking redundancy for every possible clause requires time $O(m^2)$, where $m$ is the total size of the set of clauses, an algorithm running in time $O(mn)$ exists, where $n$ is the number of variables. This running time is asymptotically different as the total size of the set of clauses $m$ can be quadratically larger than the number of variables $n$.

A number of problems that are NP-hard for the general case turned out to be polynomial for the 2CNF case. In particular, this is the case for the problem of checking redundancy, checking whether a set is an I.E.S. of another one, checking whether a clause is in all I.E.S.'s of a given formula, and checking whether a formula has an unique I.E.S. In particular, the two latter problems are polynomial thanks to previous results about I.E.S.'s: a clause is in all I.E.S.'s if and only if its is not redundant and a formula has an unique I.E.S. if and only if the set of its irredundant clauses is equivalent to it [17]. The problems of I.E.S. size and presence of a clause in an I.E.S. required a technical analysis. This required a separate study for the restrictions to formulae that are either inconsistent, consistent and implying some literals, or consistent and not implying literals. Surprisingly, these restrictions have not influenced the final results: these problems are polynomial or NP-complete depending on whether the formula is acyclic or cyclic but not on the aforementioned restriction.

One problem is still left open: that of checking the size of a I.E.S. for cyclic inconsistent formulae. Based on the other results, it seems logical to assume that this problem is NP-complete. However, considering that each case required a separate analysis, it is still possible that this case is different from the other one, and turns out to be polynomial instead.

## Appendix A. Proofs

### A.1. Unit propagation

**Lemma 1.** *For any 2CNF formula $\Pi$ and two literals $l_1$ and $l_2$, if $\Pi \models_R l_1 \vee l_2$, then $\Pi \cup \{\neg l_1\} \models_{UP} l_2$.*

**Proof.** Since $\Pi \models_R l_1 \vee l_2$, there exists a resolution tree rooted with $l_1 \vee l_2$. We prove the claim by induction on the height of the tree. The base case of recursion is when the tree is a leaf. In this case, $l_1 \vee l_2 \in \Pi$, which implies that $\Pi \cup \{\neg l_1\} \models_{UP} l_2$.

Let us therefore assume that the claim holds for any binary clause that can be proved with a tree of height $k$, and prove it for clauses requiring trees of height $k + 1$. Let us therefore consider a tree of height $k + 1$ with $l_1 \vee l_2$ in the root. Its subtrees have height less than or equal to $k$, and their roots are labeled with $l_1 \vee l_3$ and $\neg l_3 \vee l_2$ for some literal $l_3$; note that resolution does not allow to derive $l_1 \vee l_2$ from $l_1$ or from $l_2$. Fig. A.1 shows such a tree.

Since the resolution trees of $l_1 \vee l_3$ and $\neg l_3 \vee l_2$ have both height less than or equal to $k$, by the induction hypothesis both $\Pi \cup \{\neg l_1\} \models_{UP} l_3$ and $\Pi \cup \{l_3\} \models_{UP} l_2$ hold. As a result, $\Pi \cup \{\neg l_1\} \models_{UP} l_2$. □

**Lemma 2.** *For any 2CNF formula $\Pi$ and literal $l$, if $\Pi \models_R l$ then $\Pi \cup \{\neg l\} \models_{UP} \bot$.*

**Proof.** The claim is proved by induction on the height of the resolution tree rooted with $l$. In the base case, this tree is a leaf; therefore, $l \in \Pi$ holds. The claim $\Pi \cup \{\neg l\} \models_{UP} \bot$ therefore holds.

We now assume that the claim is true for any literal that is derivable from $\Pi$ using a resolution tree of height less than or equal to $k$ and prove that the same holds for height $k + 1$. Let $l$ be the root of a resolution tree of height $k + 1$. The root of this tree is $l$, and its children can be either both binary, or a unary clause resolved with a binary clause. Let us consider this latter case, depicted in Fig. A.2, first.

By induction, $\Pi \cup \{\neg l_1\} \models_{UP} \bot$, as the subtree rooted with $l_1$ have height less than or equal to $k$. By Lemma 1, $\neg l$ implies $\neg l_1$ by unit propagation. As a result, $\Pi \cup \{\neg l\} \models_{UP} \bot$.

Let us now consider the situation in which the children of $l$ are both binary clauses. Let us call $l_1$ the literal they are resolved upon, that is, the two clauses are $l \vee l_1$ and $l \vee \neg l_1$. Such a tree is depicted in Fig. A.3.

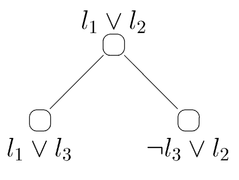

Fig. A.1. A resolution tree rooted by $l_1 \vee l_2$.

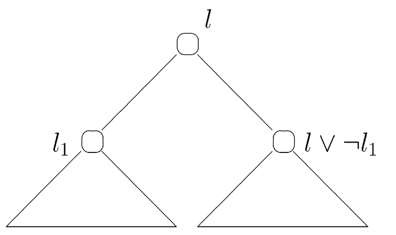

Fig. A.2. A resolution tree where a child of the root is unary.

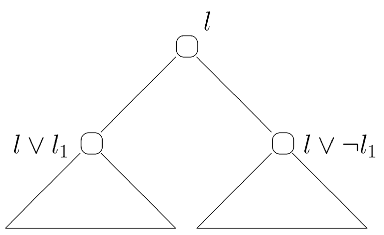

Fig. A.3. A resolution tree where the children of the root are both binary.

Since $l \vee l_1$ is the root of a resolution tree whose leaves are clauses of $\Pi$, it follows that $\Pi \models_R l \vee l_1$. By Lemma 1, $\Pi \cup \{\neg l\} \models_{UP} l_1$ and $\Pi \cup \{\neg l\} \models_{UP} \neg l_1$. Since $\neg l$ allows deriving a pair of contradictory literals by unit propagation, we have $\Pi \cup \{\neg l\} \models_{UP} \bot$. □

**Lemma 3.** *A 2CNF formula $\Pi$ is inconsistent if and only if there exists a variable $x$ such that $\Pi \cup \{x\} \models_{UP} \bot$ and $\Pi \cup \{\neg x\} \models_{UP} \bot$.*

**Proof.** The "if" direction is obvious, thanks to the soundness of unit propagation.

Let us consider a minimal regular resolution tree for $\Pi$. Its root is marked with $\bot$, so its children have to be marked $x$ and $\neg x$ for some variable $x$. Therefore, we have that $\Pi \models_R x$ and $\Pi \models_R \neg x$. By Lemma 2, the claim is proved. □

**Lemma 4.** *For every 2CNF formula $\Pi$ and literals $l_1$ and $l_2$, $\Pi \models l_1 \vee l_2$ if and only if $\Pi$ is inconsistent or $\Pi \cup \{\neg l_1\} \models_{UP} \bot$ or $\Pi \cup \{\neg l_2\} \models_{UP} \bot$ or $\Pi \cup \{\neg l_1\} \models_{UP} l_2$.*

**Proof.** The "if" direction is due to the fact the unit propagation is a sound (but not complete) entailment method, that is, if $\Pi \models_{UP} \gamma$ then $\Pi \models \gamma$.

The other direction is proved by applying Property 3. Indeed, $\Pi \models \gamma$ implies that $\Pi$ either is inconsistent or implies either $l_1$, $l_2$, or $l_1 \vee l_2$ by resolution. In turns, $\Pi \models_R l_1$ implies $\Pi \cup \{\neg l_1\} \models_{UP} \bot$, and similarly for $\Pi \models_R l_2$ thanks to Lemma 2. Moreover, $\Pi \models_R l_1 \vee l_2$ implies $\Pi \cup \{\neg l_1\} \models_{UP} l_2$ thanks to Lemma 1. □

*A.2. Inconsistent 2CNF formulae*

**Theorem 1.** *Every inconsistent set of binary clauses is cyclic.*

**Proof.** Let $\Pi$ be a set of binary clauses. By Lemma 3, if $\Pi$ is inconsistent then there exists a variable $x$ such that $\Pi \cup \{x\} \models_{UP} \bot$ and $\Pi \cup \{\neg x\} \models_{UP} \bot$. Since $\Pi$ contains only binary clauses by assumption, these two conditions are equivalent to the existence of four paths of clauses: one from $x$ to a variable $y$, one from $x$ to $\neg y$, one from $\neg x$ to a variable $z$, and one from $\neg x$ to $\neg z$. The second and fourth of these two paths can be reversed, thus leading to a path from $y$ to $\neg x$ and one from $z$ to $x$, respectively. These two paths, together with the first and third original ones, form a cycle from $x$ to $y$ to $\neg x$ to $z$ to $x$. □

**Lemma 6.** *Every 2CNF formula $\Pi$ containing a unit clauses $l$ is equivalent to $\Pi \backslash \{l\} \cup \{l \vee y, l \vee \neg y\}$, where $y$ is a new variable; this second formula is cyclic only if $\Pi$ is either cyclic or inconsistent.*

**Proof.** Equivalence is obvious. Let us assume that $\Pi$ is acyclic while $\Pi' = \Pi \backslash \{l\} \cup \{l \vee y, l \vee \neg y\}$ is not, and prove that $\Pi$ is inconsistent. Since $\Pi$ is acyclic while $\Pi'$ is not, every cycle of $\Pi'$ contains either $l \vee y$ or $l \vee \neg y$. Let us assume that one such cycle contains $l \vee y$. Since $y$ is a new variable, the only possible next clause of the cycle is $\neg y \vee l$. This means that the rest of the cycle is composed of clauses $\neg l \vee z, \ldots, w \vee \neg l$. This means that $\neg l$ is obtained by applying unit propagation from $l$ on the clauses of $\Pi \backslash \{l\}$. Since $\Pi$ contains $l$, it is inconsistent. □

**Lemma 7.** *Every resolution tree rooted with $\bot$ of a 2CNF formula has at most two unary clauses in the leaves.*

**Proof.** We prove that each subtree of the root contains at most a unary clause in the leaves. Since the root is labeled with $\bot$, its subtrees are rooted by $x$ and $\neg x$. We prove the claim for the first subtree; the proof for the second is similar.

If this subtree is only composed of $x$, the claim is proved. Otherwise, $x$ has two children, of which at most one can be unary. Since a binary clause can only be obtained by resolving two binary clauses, all its descendants are binary clauses. As a result, all binary clauses that are children of $x$ have all binary clauses as descendants.

If $x$ has a unary child, the same argument can be used on the subtree rooted by that child: at most one of its children is unary, and all descendants of the binary clauses in its children are labeled with binary clauses. As a result, this subtree has at most one unary leaf. The same argument applies to the other subtree, so the tree has to most two unary clauses in the leaves. □

**Lemma 8.** *Every irredundant inconsistent 2CNF formula contains at most two unary clauses.*

**Proof.** Let $\Pi$ be an irredundant inconsistent 2CNF formula. Consider one of its resolution proofs of inconsistency. By Lemma 7, this tree contains at most two unary clauses in the leaves. As a result, if $\Pi$ contained more than two unary clauses, the ones that are not leaves of that tree would be redundant. □

**Lemma 9.** *If $\Pi$ is an irredundant inconsistent 2CNF, $\Pi'$ and $\Pi''$ are the set of its unary and binary clauses, respectively, and $\Pi''$ is consistent, then $\Pi' \cup \Pi'' \models_{UP} \bot$.*

**Proof.** Since $\Pi''$ is consistent, $\Pi'$ cannot be empty. By Lemma 8, $\Pi'$ contains either one or two unary clauses.

Let us first assume that $\Pi' = \{l\}$. Since $\Pi' \cup \{l\} \models \bot$, we have $\Pi' \models \neg l$. Since $\Pi' \not\models_R \bot$, by Property 3 we have $\Pi' \models_R \neg l$. By Lemma 2, $\Pi' \cup \{l\} \models_{UP} \bot$.

Let us now assume that $\Pi' = \{l, l'\}$. Since $\Pi' \cup \{l, l'\} \models \bot$, we have that $\Pi' \models \neg l \vee \neg l'$. Since $\Pi$ is irredundant, it holds neither $\Pi' \models_R \neg l$ nor $\Pi' \models_R \neg l'$. As a result, $\Pi' \cup \{l\} \models_{UP} \neg l'$ by Lemma 4, which is equivalent to $\Pi' \cup \{l, l'\} \models_{UP} \bot$. □

### *A.3. Formulae implying literals*

**Lemma 10.** *If $\Pi$ is a CNF formula such that $\Pi \models l$, then $\Pi$ and $\Pi \backslash (\Pi|l) \cup \{l\}$ are equivalent.*

**Proof.** Let $M$ be a model of $\Pi$. By definition, $M$ satisfies all clauses of $\Pi$. Since $\Pi \models l$, this model assigns true to $l$. Since $\Pi \backslash (\Pi|l) \cup \{l\}$ only contains clauses of $\Pi$ and $l$, it is satisfied by $M$.

Vice versa, let $M$ be a model of $\Pi \backslash (\Pi|l) \cup \{l\}$. This model assigns true to $l$. Moreover, it satisfies all clauses of $\Pi$ not containing $l$. It also satisfies all clauses of $\Pi$ containing $l$ because it sets $l = \text{true}$; therefore, it satisfies all clauses of $\Pi$. □

**Lemma 11.** *Let $\Pi$ be a formula implying the literal $l$. If the clause $\gamma \in \Pi$ does not contain $l$, then $\gamma$ is redundant in $\Pi$ if and only if it is redundant in $\Pi \backslash (\Pi|l) \cup \{l\}$.*

**Proof.** Since $\gamma$ does not contain $l$, it is both in $\Pi$ and in $\Pi \backslash (\Pi|l) \cup \{l\}$. We therefore only have to prove that is entailed by $\Pi \backslash \{\gamma\}$ if and only if it is entailed by $(\Pi \backslash (\Pi|l) \cup \{l\}) \backslash \{\gamma\}$. We prove that these two formulae are equivalent. Since $\gamma$ is not a clause of $\Pi|l$, we have that $(\Pi \backslash (\Pi|l) \cup \{l\}) \backslash \{\gamma\}$ is the same as $(\Pi \backslash \{\gamma\}) \backslash (\Pi|l) \cup \{l\}$. By Lemma 10, this formula is equivalent to $\Pi \backslash \{\gamma\}$. □

**Lemma 12.** *Let $\Pi$ be a consistent 2CNF such that $\Pi \models l$. If $\Pi'$ is an* I.E.S. *of $\Pi$ then $\Pi_2 = \Pi' \backslash (\Pi|l) \cup \{l\}$ is an* I.E.S. *of $\Pi \backslash (\Pi|l) \cup \{l\}$.*

**Proof.** Since $\Pi'$ is an I.E.S. of $\Pi$, we have that $\Pi' \subseteq \Pi$. As a result, $\Pi' \backslash (\Pi|l) \cup \{l\} \subseteq \Pi \backslash (\Pi|l) \cup \{l\}$. Containment is the first condition for a formula being an I.E.S. of another formula. We now show equivalence and irredundancy.

Since $\Pi'$ is equivalent to $\Pi$, we have $\Pi' \models l$. By Lemma 10, $\Pi'$ is equivalent to $\Pi' \backslash (\Pi|l) \cup \{l\}$, which is indeed $\Pi_2$. As a result, $\Pi_2$ is equivalent to $\Pi'$, which is equivalent to $\Pi$, which is equivalent to $\Pi \backslash (\Pi|l) \cup \{l\}$ by Lemma 10.

Let us now assume that $\Pi'\backslash(\Pi|l) \cup \{l\}$ is redundant. The clause $l$ cannot be redundant because it is the only clause mentioning the literal $l$. Therefore, there exists a clause $\gamma$ not containing $l$ such that $\gamma$ is redundant in $\Pi'\backslash(\Pi|l) \cup \{l\}$. By Lemma 11, $\gamma$ is also redundant in $\Pi'$, thus contradicting the assumption that $\Pi'$ is an I.E.S. □

The converse of this lemma does not hold, as already shown by the following example.

$$\Pi = \{x \vee x_1, x \vee x_2, \neg x_1 \vee y, \neg x_2 \vee \neg y, \neg x \vee y\}$$
$$l = x$$
$$\Pi\backslash(\Pi|l) \cup \{l\} = \{\neg x_1 \vee y, \neg x_2 \vee \neg y, \neg x \vee y, x\}$$
$$\Pi_2 = \{\neg x_2 \vee \neg y, \neg x \vee y, x\}$$
$$\Pi_2\backslash\{l\} \cup (\Pi|l) = \{\neg x_2 \vee \neg y, \neg x \vee y, x \vee x_1, x \vee x_2\}$$

A possible (but incorrect) proof of the converse of Lemma 12 would go by considering that $x_1$ and $x_2$ should always entail $y$ and $\neg y$, respectively, in every I.E.S. of $\Pi$, and therefore in any I.E.S. of $\Pi\backslash(\Pi|l) \cup \{l\}$, because this formula is equivalent to $\Pi$. What makes this proof fail on the counterexample above is that $\neg x_1 \vee y$ holds also because of $x$ and $\neg x \vee y$, and this proof relies on $x$, which is removed while "coming back" from $\Pi_2$ to $\Pi\backslash\{l\} \cup (\Pi|l)$.

The problem with this proof is that the clause $\neg x_1 \vee y$, which is part of the proof of $x$, is not necessary because it can be derived from $l = x$ in $\Pi\backslash(\Pi|l) \cup \{l\}$. On the other hand, $l$ can only derive clauses of the form $l \vee$ *something*, or $y \vee$ *something* where $y$ is a consequence of $l$. In other words, what makes the proof fail is the possible entailment of clauses containing literals that are derivable from $l$. On the other hand, $\Pi \models l$; therefore, $\Pi \models y$. The counterexample therefore relies on the presence in $\Pi\backslash(\Pi|l) \cup \{l\}$ of clauses containing literals that are entailed by $\Pi$. Replacing all such clauses with that literal would therefore invalidate the counterexample. Since $\Pi_{\models}$ is the set of literals entailed by $\Pi$, the set $\Pi|\Pi_{\models}$ contains all clauses of $\Pi$ containing a literal that is entailed by $\Pi$.

**Lemma 13.** *Every clause of a 2CNF formula $\Pi$ either is in $\Pi|\Pi_{\models}$ or does not contain literals in $\Pi_{\models}$ nor their negation.*

**Proof.** Let $l \in \Pi_{\models}$. All clauses containing $l$ are in $\Pi|\Pi_{\models}$ by definition. On the other hand, all clauses containing the negation of $l$ are in the form $\neg l \vee l'$. Since $\Pi \models l$, we have $\Pi \models l'$. Therefore $l \vee l' \in \Pi|\Pi_{\models}$. □

**Lemma 14.** *Let $\Pi$ be a consistent 2CNF formula. If $\Pi_2$ is an* I.E.S. *of $\Pi\backslash(\Pi|\Pi_{\models}) \cup \Pi_{\models}$, then $\Pi_2\backslash\Pi_{\models} \cup (\Pi|\Pi_{\models})$ is equivalent to $\Pi$.*

**Proof.** Since $\Pi$ and $\Pi\backslash(\Pi|\Pi_{\models}) \cup \Pi_{\models}$ are equivalent and $\Pi_2$ is an I.E.S. of $\Pi\backslash(\Pi|\Pi_{\models}) \cup \Pi_{\models}$, we have that $\Pi_2$ and $\Pi$ are equivalent. The claim is proved by showing that $\Pi_2\backslash\Pi_{\models} \cup (\Pi|\Pi_{\models})$ entails $\Pi_{\models}$. This would prove that $\Pi_2\backslash\Pi_{\models} \cup (\Pi|\Pi_{\models})$ is equivalent to $\Pi_2\backslash\Pi_{\models} \cup (\Pi|\Pi_{\models}) \cup \Pi_{\models}$, which is a superset of $\Pi_2$ and is therefore equivalent to $\Pi$.

Intuitively, the proof is as follows: if $l \in \Pi_{\models}$, then there is a proof of $l$ in $\Pi$. This proof involves some clauses of $\Pi|\Pi_{\models}$ and some clauses in $\Pi\backslash(\Pi|\Pi_{\models})$. On the other hand, everything that is entailed in $\Pi$ is also entailed in its equivalent formula $\Pi_2\backslash(\Pi|\Pi_{\models}) \cup \Pi_{\models}$. Since $\Pi_2\backslash(\Pi|\Pi_{\models})$ and $\Pi_{\models}$ are built over disjoint literals, every clause of $\Pi$ not containing literals that are entailed by $\Pi$ is derivable in $\Pi_2\backslash(\Pi|\Pi_{\models})$.

Let us formally prove the claim. Let $l \in \Pi_{\models}$, that is, $\Pi \models l$. We show that $\Pi_2\backslash\Pi_{\models} \cup (\Pi|\Pi_{\models}) \models l$. Since $\Pi \models l$, two situations are possible: either $l \in \Pi$, or $l \notin \Pi$. In the first case, $l \in \Pi|\Pi_{\models}$, and the claim is true because $l$ is in $\Pi_2\backslash\Pi_{\models} \cup (\Pi|\Pi_{\models})$.

Let us now consider the case $l \notin \Pi$. Since $\Pi \models l$ and $\Pi$ is consistent, $\Pi \cup \{\neg l\}$ allows deriving a pair of opposite literals by unit propagation. Let the following be the chains of clauses used in the unit propagation from $\neg l$ to these pairs of opposite literals:

$$\neg p_1 \vee p_2, \quad \neg p_2 \vee p_3, \quad \ldots, \quad \neg p_{n-1} \vee p_n$$
$$\neg n_1 \vee n_2, \quad \neg n_2 \vee n_3, \quad \ldots, \quad \neg n_{m-1} \vee \neg n_m$$
$$\text{where} \quad \neg l = p_1 = n_1 \quad \text{and} \quad p_n = \neg n_m$$

Consider an clause $\neg l_i \vee l_{i+1}$ of any of these two chains. By Lemma 13, either this clause is in $\Pi|\Pi_{\models}$ or it does not share variables with $\Pi_{\models}$. In the first case, $l_i \vee l_{i+1} \in \Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})$. We show that $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models}) \models \neg l_i \vee l_{i+1}$ holds in the second case.

Since $\neg l_i \vee l_{i+1}$ is in $\Pi$, it holds $\Pi_2 \models \neg l_i \vee l_{i+1}$. On the other hand, $\Pi_2 = (\Pi_2 \backslash \Pi_{\models}) \cup (\Pi_2 \cap \Pi_{\models})$. These two parts of $\Pi_2$ contain disjoint literals because of Corollary 2. Since $\Pi_2 \models \neg l_i \vee l_{i+1}$ and neither $l_i$, $l_{i+1}$, nor their negations are in $\Pi_{\models}$, then $\Pi_2 \backslash \Pi_{\models} \models \neg l_i \vee l_{i+1}$ because the other part of $\Pi_2$ contains literals that are mentioned neither in $\Pi_2 \backslash \Pi_{\models}$ nor in $\neg l_i \vee l_{i+1}$.

Of the clauses of the two chains above, therefore, we have that a clause $\neg l_i \vee l_{i+1}$ is either in $\Pi|\Pi_{\models}$ or is entailed by $\Pi_2 \backslash \Pi_{\models}$. As a result, $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})$ entails all these clauses; therefore, unit propagation of $\neg l$ in this formula leads to a pair of opposite literals. □

**Lemma 15.** *Let $\Pi$ be a consistent 2CNF formula. If $\Pi_2$ is an* I.E.S. *of $\Pi \backslash (\Pi|\Pi_{\models}) \cup \Pi_{\models}$, then there exists $\Pi_1 \subseteq \Pi|\Pi_{\models}$ such that $\Pi_1 \cup (\Pi_2 \backslash \Pi_{\models})$ is an* I.E.S. *of $\Pi$.*

**Proof.** Lemma 14 shows that adding $\Pi|\Pi_{\models}$ to $\Pi_2 \backslash \Pi_{\models}$ results in a set that is equivalent to $\Pi$. We are now trying to prove that an I.E.S. of $\Pi$ can be obtained from $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})$ without removing any clause of $\Pi_2 \backslash \Pi_{\models}$. What we actually prove is that no clause of $\Pi_2 \backslash \Pi_{\models}$ is redundant in $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})$, thus proving that all I.E.S.'s of $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})$ contain all clauses of $\Pi_2 \backslash \Pi_{\models}$ by Property 1.

Let $\gamma \in \Pi_2 \backslash \Pi_{\models}$. Assume that $\gamma$ is redundant in $\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})$, that is:

$$\big(\Pi_2 \backslash \Pi_{\models} \cup (\Pi|\Pi_{\models})\big) \backslash \{\gamma\} \models \gamma$$

Since all clauses in $\Pi|\Pi_{\models}$ contain a literal in $\Pi_{\models}$ by definition, we have that $\Pi_{\models} \models \Pi|\Pi_{\models}$. As a result, $\Pi_{\models}$ is logically stronger than $\Pi|\Pi_{\models}$, and the above formula therefore implies:

$$(\Pi_2 \backslash \Pi_{\models} \cup \Pi_{\models}) \backslash \{\gamma\} \models \gamma$$

This is the same as $\Pi_2 \backslash \{\gamma\} \models \gamma$, contradicting the assumption that $\Pi_2$ is irredundant. □

**Lemma 16.** *Every consistent 2CNF formula $\Pi$ implying a literal $l$ and containing three or more clauses containing $l$ contains at least a redundant clause containing $l$.*

**Proof.** If $l \in \Pi$, any other clause containing $l$ is redundant. Let us assume $l \notin \Pi$. Since $\Pi$ is consistent, $\Pi \models l$ is equivalent to $\Pi \cup \{\neg l\} \models_{UP} \bot$. By definition, this formula is true if and only if $\neg l$ allows deriving a pair of complementary literals $x$ and $\neg x$ by unit propagation.

Let us first assume that neither $l$ is neither $x$ nor $\neg x$. Consider the sequences of clauses used in the derivation of $x$ and $\neg x$ from $l$. Without loss of generality, we can assume that $l$ is only contained in the first clauses of these two sequences.

Removing all clauses containing $l$ but the first clause in each of these two sequences, we obtain a formula that still entails $l$, and therefore allows deriving all clauses that have been removed, contradicting the assumption that $\Pi$ is irredundant.

A similar proof can be used for the case in which $l = x$ or $l = \neg x$. In this case, the first clause of the sequence of clauses used in the derivation of $\neg l$ from $l$ allows deriving all other clauses containing $l$. □

*A.4. Acyclic consistent 2CNF formulae not implying literals have a single* I.E.S.

**Lemma 17.** *If $\Pi$ is a consistent acyclic 2CNF and $\Pi \cup \{l\} \models_{UP} l'$, then $\Pi' \cup \{l\} \models_{UP} l'$ holds for every* I.E.S. *$\Pi'$ of $\Pi$.*

**Proof.** Since $\Pi \cup \{l\} \models_{UP} l'$, it holds that $\Pi \models \neg l \vee l'$, and the same therefore holds for $\Pi'$ because this formula is equivalent to $\Pi$. On the other hand, $\Pi'$ is consistent and does not entail literals because it is equivalent to $\Pi$. As a result, $\Pi' \models \neg l \vee l'$ is equivalent to $\Pi' \cup \{l\} \models_{UP} l'$. □

The next two lemmas prove Corollary 4.

**Lemma 23.** *If $\Pi$ is a consistent acyclic 2CNF formula not implying literals and $l_1 \in M_\Pi(l)$ then every* I.E.S. *of $\Pi$ contains $\neg l \vee l_1$.*

**Proof.** Let us assume that $\Pi'$ is an I.E.S. of $\Pi$. Since $\Pi \cup \{l\} \models_{UP} l_1$, we have that $\Pi' \cup \{l\} \models_{UP} l_1$ holds by Lemma 17. Let us assume that $\neg l \vee l_1$ is not in $\Pi'$. Then, the path of clauses from $l$ to $l_1$ is made of a clause $\neg l \vee l_2$ with $l_2 \neq l_1$ followed by a path from $l_2$ to $l_1$. This path cannot include $l$ because otherwise $\Pi$ would be cyclic. Therefore, this path from $l_2$ to $l_1$ is all contained in $R_\Pi(l)$. This is however in contradiction with $l_1$ being in $M_\Pi(l)$. □

The second lemma is the converse of the previous one, stating that no I.E.S. contains a clause made of a literal not in $C_\Pi(l)$ and $l$.

**Lemma 24.** *If $l_1 \in C_\Pi(l) \backslash M_\Pi(l)$ then no* I.E.S. *of the consistent acyclic 2CNF formula $\Pi$ contains $\neg l \vee l_1$.*

**Proof.** By definition of $M_\Pi(l)$, if a literal $l_1$ is in $C_\Pi(l)$ but not in $M_\Pi(l)$, then there is a literal $l'$ in $M_\Pi(l)$ such that $R_\Pi(l)$ contains a path from $l'$ to $l_1$. Let $\Pi'$ be a I.E.S. of $\Pi$. Since $l_1$ is reachable from $l'$, it holds that $\Pi \cup \{\neg l'\} \models_{UP} l_1$ and, therefore, $\Pi' \cup \{\neg l'\} \models_{UP} l_1$.

By the previous lemma, $\Pi'$ contains $\neg l \vee l'$. As a result, $\Pi' \cup \{l\} \models_{UP} l'$. Since $\Pi' \cup \{\neg l'\} \models_{UP} l_1$, we have that $l_1$ can be reached from $l$ by first using the clause $\neg l \vee l'$. Since the formula is acyclic, the clauses used in the derivation $\Pi' \cup \{\neg l'\} \models_{UP} l_1$ do not contain $l$. As a result, unit propagation allows to reach $l_1$ from $l$ without using the clause $\neg l \vee l_1$. In other words, $\Pi' \backslash \{\neg l \vee l_1\} \cup \{l\} \models_{UP} l_1$, thus proving that $\neg l \vee l_1$ is redundant in $\Pi'$, contradicting the assumption that $\Pi'$ is an I.E.S. □

We can then conclude that $M_\Pi(l)$ exactly identifies all clauses of $D_\Pi(l)$ that are in an I.E.S. of $\Pi$, as stated by Corollary 4.

## *A.5. Implied literals not in a cycle of a consistent 2CNF formula*

**Lemma 18.** *If $\Pi'$ is an* I.E.S. *of a consistent acyclic 2CNF formula $\Pi$ such that $\Pi \models \neg l$ and $l$ is not in any cycle of clauses of $\Pi$, then the following are all* I.E.S.*'s of $\Pi$:*

(1) $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_1\}$ *with* $l_1 \in S_\Pi(l)$;
(2) $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_2, \neg l \vee l_3\}$ *with* $(l_2, l_3) \in P_\Pi(l)$.

**Proof.** $\Pi' \cap R_\Pi(l)$ contains all clauses of $\Pi'$ but those containing $\neg l$. As a result, if we can prove that the formulae above entail $\neg l$, that would prove that they are equivalent to $\Pi'$.

Let us consider $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_1\}$ first. Since $l_1 \in S_\Pi(l)$, we have that $\Pi \cup \{l_1\} \models_{UP} \bot$. Since $l$ is not part of a cycle, all derivations of $\bot$ from $l_1$ are entirely contained in $R_\Pi(l)$. Since $\Pi'$ is equivalent to $\Pi$, it holds $\Pi' \cup \{l_1\} \models \bot$; since $\Pi'$ is a subset of $\Pi$, all derivations of $\bot$ from $l_1$ only use clauses of $R_\Pi(l)$. As a result, $(\Pi' \cap R_\Pi(l)) \cup \{l_1\} \models \bot$, which implies that $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_1\} \cup \{l\} \models \bot$.

Since $\Pi'$ is irredundant, $\Pi' \cap R_\Pi(l)$ is irredundant as well. In order to prove that $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_1\}$ is irredundant, observe that $\neg l \vee l_1$ is not redundant, as $\Pi' \cap R_\Pi(l)$ does not contain clauses containing $\neg l$ and does not therefore entail $\neg l$. Regarding the clauses of $\Pi' \cap R_\Pi(l)$, since they do not contain $\neg l$ by definition, Lemma 11 applies: they are redundant in $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_1\}$ if and only if they are redundant in $\Pi' \cap R_\Pi(l)$, which is impossible because $\Pi'$ is irredundant.

The proof for $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_2, \neg l \vee l_3\}$ with $(l_2, l_3) \in P_\Pi(l)$ is similar. This formula implies $\neg l$ because $\neg l_3$ is reachable from $l_2$, and all paths from $l_2$ to $\neg l_3$ are in $R_\Pi(l)$. Since $\Pi'$ is equivalent to $\Pi$, it contains one such path, which is therefore entirely contained in $\Pi' \cap R_\Pi(l)$. As a result, the addition of the clauses $\neg l \vee l_2$ and $\neg l \vee l_3$ allows the entailment of $\neg l$.

The proof of irredundancy of $(\Pi' \cap R_\Pi(l)) \cup \{\neg l \vee l_2, \neg l \vee l_3\}$ is also similar to the previous one. However, for this proof to work we also need the fact that neither $l_2$ nor $l_3$ are in $S_\Pi(l)$, and therefore one of them is not sufficient for entailing $\neg l$. □

## *A.6. Redundancy checking: Inconsistent 2CNF formulae*

**Lemma 19.** *A 2CNF formula $\Pi$ is inconsistent and irredundant if and only if it is composed of two simple chains of clauses like the following ones*:

$$x \vee l_1, \quad \neg l_1 \vee l_2, \quad \ldots, \quad \neg l_m \vee y, \quad \neg y \vee s_1, \quad \ldots, \quad \neg s_m \vee \neg y$$
$$\neg x \vee p_1, \quad \neg p_1 \vee p_2, \quad \ldots, \quad \neg p_m \vee z, \quad \neg z \vee q_1, \quad \ldots, \quad \neg q_m \vee \neg z$$

**Proof.** If $\Pi$ is inconsistent, by Lemma 3 there exists a variable $x$ such that $\Pi \cup \{x\} \models_{UP} \bot$ and $\Pi \cup \{\neg x\} \models_{UP} \bot$. In turn, $\Pi \cup \{x\} \models_{UP} \bot$ implies the existence of a chain allowing the derivation of $y$ and $\neg y$ from $x$ by unit propagation, as explained in Section 3.2. The same holds for $\Pi \cup \{\neg x\} \models_{UP} \bot$. The clauses of these two chains imply inconsistency. Therefore, if $\Pi$ contains other clauses, they are redundant. □

## *A.7. Presence in an* I.E.S.*: Cyclic inconsistent 2CNF formulae*

**Lemma 21.** *The problem of checking in a graph the existence of a simple path from node $x$ to node $y$ including a given edge is* NP*-complete.*

**Proof.** Membership is obvious. Hardness is proved by reduction from the problem *path via a node*: given a graph $G$ and three nodes $x$, $y$, and $m$, decide whether there exists a simple path from $x$ to $y$ including the node $m$. This problem is NP-complete [15].

The reduction is as follows: replace the node $m$ with two nodes $a$ and $b$ joined by an edge. Replace every edge $n \rightarrow m$ with $n \rightarrow a$ and every edge $m \rightarrow n$, with $b \rightarrow n$, as shown in Fig. A.4.

By construction, every simple path containing the node $m$ in the original graph contains an edge that is incoming to $m$ and is outgoing from $m$. This is possible if and only if the new graph has a simple path including the edge $m_1 \rightarrow m_2$. □

## *A.8. Presence in an* I.E.S.*: Clause not in a cycle*

It has already been proved that the problem of presence of a clause in an I.E.S. of acyclic consistent 2CNF formulae is polynomial. In this section, we extend this result to the case in which cycles are allowed, but none include the clause to check.

Acyclic consistent 2CNF formulae not implying single literals have a unique I.E.S., which can be determined in polynomial time; this result implies that checking the presence of a clause in an I.E.S. is easy. We extend this result to the case in which the formula contains some cycles, but none include the clause under consideration.

Fig. A.5 explains the idea of the proof: if we box all literals that are equivalent to $l_1$ and all literals that are equivalent to $l_2$, then $\neg l_1 \vee l_2$ is in some I.E.S. if and only if all paths from the $l_1$ box to the $l_2$ box only contain nodes in the boxes. In this case, removing all edges from the $l_1$ box to the $l_2$ box, the clause $\neg l_1 \vee l_2$ is the only one that makes $l_2$ reachable from $l_1$. This clause is therefore necessary. On the other hand, if there is a path from $l_1$ to $l_2$ that includes a node $l$ not in the boxes, then every I.E.S. includes a path from the $l_1$ box to $l$ and from $l$ to the $l_2$ box because such paths exist in the original formula. As a result, $\neg l_1 \vee l_2$ is always redundant.

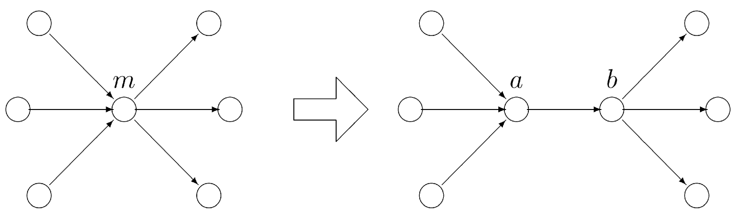

Fig. A.4. Replacement of a node by two nodes and an edge.

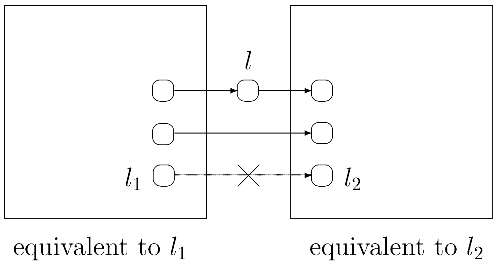

Fig. A.5. The presence of the edges joining $l$ makes the clause from $\neg l_1 \vee l_2$ redundant.

**Theorem 11.** *If $\Pi \not\models l_1 \equiv l_2$, then $\neg l_1 \vee l_2$ is in a* I.E.S. *of $\Pi$ if and only if all paths from $l_1$ to $l_2$ contain only literals that $\Pi$ makes equivalent either to $l_1$ or to $l_2$.*

**Proof.** Let us assume that all paths from $l_1$ to $l_2$ only contains literals that are equivalent to either $l_1$ or to $l_2$. Since $l_1$ and $l_2$ are not equivalent, every such path can be written as $(l_1, \ldots, l_3, l_4, \ldots, l_2)$, where $\Pi$ makes literals $l_1, \ldots, l_3$ equivalent and literals $l_4, \ldots, l_2$ equivalent.

Let us now remove the clause $\neg l_3 \vee l_4$. By assumption, $l_4$ can still be reached from $l_3$ by first going to $l_1$, then to $l_2$, and then to $l_4$. Iterating this process over all paths from $l_1$ to $l_2$, we end up with a set of clauses whose only path from $l_1$ to $l_2$ is the single clause $\neg l_1 \vee l_2$. This clause is now irredundant: removing other redundant clauses, we obtain a I.E.S. containing $\neg l_1 \vee l_2$.

Let us now assume the converse: there is a path from $l_1$ to $l_2$ that contains some literals that are not equivalent to $l_1$ nor to $l_2$. Such a path can be written as: $(l_1, \ldots, l_3, l_4, \ldots, l_5, l_6, \ldots, l_2)$, where $l_1, \ldots, l_3$ are equivalent to each other, as are $l_6, \ldots, l_2$, but none of these literals are equivalent to any literal in $l_4, \ldots, l_5$. We prove that no I.E.S. contains the clause $\neg l_1 \vee l_2$.

Let $\Pi'$ be an equivalent subset of $\Pi$. Being equivalent to $\Pi$, it must contain a number of cycles that make all literals $l_1, \ldots, l_3$ equivalent to each other and all literals $l_6, \ldots, l_2$ equivalent to each other. Note that $\neg l_1 \vee l_2$ cannot be in one of such cycles; otherwise, $\Pi \models l_1 \equiv l_2$.

Since $\Pi \models \neg l_1 \vee l_4$ and $\Pi \models \neg l_4 \vee l_2$, the set $\Pi'$ must contain a path from $l_1$ to $l_4$ and a path from $l_4$ to $l_2$. If the first path includes $\neg l_1 \vee l_2$, then we would have a path from $l_2$ to $l_4$. Since $\Pi$ contains a path from $l_4$ to $l_2$, then $l_4$ and $l_2$ would be equivalent. For the same reason, the path from $l_4$ to $l_2$ does not contain the clause $\neg l_1 \vee l_2$.

We have therefore proved that $\Pi'$ includes some sets of clauses not containing $\neg l_1 \vee l_2$ but allow to conclude that $l_1, \ldots, l_3$ are equivalent to each other, that $l_6, \ldots, l_2$ are equivalent to each other, that $l_1$ implies $l_4$, and that $l_4$ implies $l_2$. The clause $\neg l_1 \vee l_2$ is therefore redundant. $\square$

This theorem implies that checking whether $\neg l_1 \vee l_2$ is in some I.E.S. is easy if $l_1$ is not equivalent to $l_2$. Indeed, the set of nodes in the paths from $l_1$ to $l_2$ can be found by intersecting the sets of nodes that are reached from $l_1$ and the ones $l_2$ is reachable from. The clause $\neg l_1 \vee l_2$ is redundant if and only if this intersection contains a literal that is not equivalent neither to $l_1$ nor to $l_2$.

We now consider the case where a literal of the clause to be checked for presence in an I.E.S. is entailed by the formula.

The I.E.S.'s of a consistent acyclic 2CNF formula can be compactly expressed as a number of independent choices. This makes the problem of checking the presence of a clause in an I.E.S. polynomial. We extend this result to the case in which a literal of the clause to check is implied by the formula but the clause is not contained in any cycle of clauses. The problem has already been proved to be NP-complete if the clause is contained in a cycle.

**Theorem 12.** *The problem of deciding whether $\neg l_1 \vee l_2$ is in an* I.E.S. *of a consistent 2CNF formula $\Pi$ is polynomial if $\Pi \models \neg l_1$ and $\neg l_1 \vee l_2$ is not in any cycle of clauses.*

**Proof.** The proof is similar to that of Lemma 18 with $CC_\Pi(l_1)$ in place of $l_1$ alone. The set $CC_\Pi(l_1)$ of literals that are in a cycle with $l_1$ can be determined in polynomial time. The clause $\neg l_1 \vee l_2$ is in an I.E.S. of $\Pi$ if and only if either $\Pi \cup \{l_2\} \models_{UP} \bot$ or $\Pi \cup \{l_2\} \models_{UP} \neg l_3$, where $l_3 \in CC_\Pi(l)$.

This algorithm is correct because there is no path from $l_2$ to $l_1$. Therefore, there is no path from $l_2$ to any node of $CC_\Pi(l)$. As a result, unit propagation from $l_2$ to $\bot$ or to $\neg l_3$ do not generate any literal in $CC_\Pi(l_1)$.

Let $\Pi'$ be an I.E.S. of $\Pi$. The derivations from $l_2$ to $\bot$ or to $\neg l_3$ still hold in $\Pi'$ and do not involve literals in $CC_\Pi(l_1)$. If $\bot$ is derivable from $l_2$, we can obtain an I.E.S. by removing all clauses $\neg l \vee l'$ with $l \in CC_\Pi(l_1)$ and adding $\neg l_1 \vee l_2$ and a minimal number of clauses to make $l_2$ reachable from any literal in $CC_\Pi(l_1)$. This addition makes the negation of all literals of $CC_\Pi(l_1)$ entailed. This is therefore an I.E.S. of $\Pi$ because all clauses that have been removed contain the negation of a literal in $CC_\Pi(l_1)$.

A similar proof can be given for the case $\Pi \cup \{l_2\} \models_{UP} \neg l_3$. □

## Appendix B. Notation

$$C_\Pi(l) = \{l' \mid \neg l \vee l' \in \Pi\}$$

$$D_\Pi(l) = \{\gamma \in \Pi \mid \gamma = \neg l \vee l'\}$$

$$R_\Pi(l) = \Pi \backslash D_\Pi(l)$$

$$M_\Pi(l) = \{l' \in C_\Pi(l) \mid \nexists l'' \in C_\Pi(l) \text{ such that } R_\Pi(l) \cup \{l''\} \models_{UP} l'\}$$

$$S_\Pi(l) = \{l' \in C_\Pi(l) \mid R_\Pi(l) \cup \{l'\} \models_{UP} \bot\}$$

$$P_\Pi(l) = \{(l_1, l_2) \mid l_1, l_2 \in C_\Pi(l) \text{ and } R_\Pi(l) \cup \{l_1\} \models_{UP} \neg l_2\} \backslash S_\Pi(l)$$

$$CC_\Pi(l) = \{l' \mid l \text{ and } l' \text{ are in a cycle}\}$$

## References

[1] G. Ausiello, A. D'Atri, D. Saccà, Minimal representation of directed hypergraphs, SIAM Journal on Computing 15 (2) (1986) 418–431.

[2] R. Bruni, Approximating minimal unsatisfiable subformulae by means of adaptive core search, Discrete Applied Mathematics 130 (2) (2003) 85–100.

[3] H. Büning, X. Zhao, Extension and equivalence problems for clause minimal formulae, Annals of Mathematics and Artificial Intelligence 43 (1) (2005) 295–306.

[4] J. Buresh-Oppenheim, D. Mitchell, Minimum witnesses for unsatisfiable 2CNFs, in: Proceedings of the Ninth International Conference on Theory and Applications of Satisfiability Testing (SAT 2006), 2006, pp. 42–47.

[5] A. del Val, On 2-SAT and renamable Horn, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000), 2000, pp. 279–284.

[6] S. Even, A. Itai, A. Shamir, On the complexity of timetable and multicommodity flow problems, SIAM Journal on Computing 5 (4) (1976) 691–703.

[7] H. Fleischner, O. Kullmann, S. Szeider, Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference, Theoretical Computer Science 289 (2002) 503–516.

[8] S. Fortune, J. Hopcroft, J. Wyllie, The directed subgraph homeomorphism problem, Theoretical Computer Science 10 (2) (1980) 111–121.

[9] A. Ginsberg, Knowledge base reduction: A new approach to checking knowledge bases for inconsistency & redundancy, in: Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI'88), 1988, pp. 585–589.

[10] G. Gottlob, C.G. Fermüller, Removing redundancy from a clause, Artificial Intelligence 61 (1993) 263–289.

[11] P. Hammer, A. Kogan, Optimal compression of propositional Horn knowledge bases: Complexity and approximation, Artificial Intelligence 64 (1) (1993) 131–145.

[12] E. Hemaspaandra, G. Wechsung, The minimization problem for Boolean formulas, in: Proceedings of the Thirty-eighth Annual Symposium on the Foundations of Computer Science (FOCS'97), 1997, pp. 575–584.

[13] D. Kavvadias, M. Sideri, E. Stavropoulos, Generating all maximal models of a Boolean expression, Information Processing Letters 74 (2000) 157–162.

[14] S. Khuller, B. Raghavachari, M. Fellows, Approximating the minimum equivalent digraph, SIAM Journal on Computing 24 (4) (1995) 859–872.

[15] A. LaPaugh, C. Papadimitriou, The even-path problem for graphs and digraphs, Networks 14 (1984) 507–513.

[16] P. Liberatore, Merging locally correct knowledge bases: A preliminary report, Tech. Rep. cs.AI/0212053, Computing Research Repository (CoRR), 2002.

[17] P. Liberatore, Redundancy in logic I: CNF propositional formulae, Artificial Intelligence 163 (2) (2005) 203–232.

[18] P. Liberatore, Redundancy in logic III: Non-monotonic reasoning. Tech. Rep. cs.LO/0507048, Computing Research Repository (CoRR), 2005.

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

[20] A. Meyer, L. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: Proceedings of the Thirteenth Annual Symposium on Switching and Automata Theory (FOCS'72), 1972, pp. 125–129.
[21] C. Papadimitriou, D. Wolfe, The complexity of facets resolved, Journal of Computer and System Sciences 37 (1988) 2–13.
[22] W.V. Quine, On cores and prime implicants of truth functions, American Mathematical Monthly 66 (1959) 755–760.
[23] N. Robertson, P. Seymour, Graph minors. XX. Wagner's conjecture, Journal of Combinatorial Theory, Series B 92 (2004) 325–357.
[24] S. Sahni, Computationally related problems, SIAM Journal on Computing 3 (4) (1974) 262–279.
[25] J. Schmolze, W. Snyder, Detecting redundant production rules, in: Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97), 1997, pp. 417–423.
[26] H. Shen, H. Zhang, Improving exact algorithms for MAX-2-SAT, Annals of Mathematics and Artificial Intelligence 44 (4) (2005) 419–436.
[27] P. Stuckey, L. Zheng, Improving GSAT using 2SAT, in: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP 2002), 2002, pp. 691–695.
[28] C. Umans, The minimum equivalent DNF problem and shortest implicants, in: Proceedings of the Thirty-Ninth Annual Symposium on the Foundations of Computer Science (FOCS'98), 1998, pp. 556–563.
[29] X. Zhao, D. Ding, Complexity results for 2cnf default theories, Fundamenta Informaticae 45 (4) (2001) 393–404.