About Some UP-Based Polynomial Fragments of SAT

Mohammad Al-Saedi[1,2]
(1) Univ. of Mustansiriyah
Baghdad, Irak
balasim2001@yahoo.com

Éric Grégoire[2] **Bertrand Mazure**[2] **Lakhdar Saïs**[2]
(2) CRIL
Université d'Artois & CNRS
rue Jean Souvraz SP18, F-62307 Lens, France
{alsaedi,gregoire,mazure,sais}@cril.univ-artois.fr

Abstract

In this paper, we extend one Tovey's polynomial fragment of SAT through the use of the unit propagation mechanism. We also answer an open question about connections between the UP-Horn class (and other UP-based polynomial variants) and Dalal's polynomial Quad class, for which unit propagation is a cornerstone.

Introduction

These last two decades, propositional reasoning and search has been a very active topic of research in A.I. Thanks to a dramatic improvement in the time-efficiency of satisfiability checking procedures on many instances (see e.g., (SAT-Competition 2013)), the Boolean framework is now widely recognized as a powerful setting for many reasoning and A.I. problem-solving paradigms. SAT, namely checking whether a set of propositional clauses is satisfiable or not, has also attracted much attention for a very long time in theoretical computer science since it is a canonical NP-complete problem (Cook 1971); as such, SAT is thus expected to remain intractable in the worst case unless P=NP.

In this paper, the focus is on some polynomial fragments of SAT. We investigate open issues about the use of the linear-time unit propagation deduction mechanism (in short, UP) to extend some fragments of SAT so that they can still be recognized and solved in polynomial time.

A first result of this paper is a family of extensions of one well-known Tovey's polynomial fragment (Tovey 1984) to also include instances that can be simplified using UP. Then, we compare two existing polynomial fragments based on UP: namely, Quad (Dalal 1996) and UP-Horn (Fourdrinoy et al. 2007). Interestingly, many benchmarks from the DIMACS repository (DIMACS 1993) and from SAT competitions (SAT-Competition 2013) have been shown to belong to UP-Horn (Fourdrinoy et al. 2007). We answer an open question about the connections between these two classes: we show that UP-Horn and some other UP-based variants are strict subclasses of $\bigcup$Quad, where $\bigcup$Quad is the union of all Quad classes obtained by investigating all possible orderings of clauses.

The paper is organized as follows. In the next section, we provide some preliminaries about SAT and unit propagation. In section 3, we recall the main three fragments of SAT that we investigate in this paper: namely, one of the Tovey's fragments, Dalal's Quad and UP-Horn. In section 4, we extend this latter Tovey's class in several directions using UP. In section 6, we show connections between Quad and UP-Horn (and other UP-variants). In section 7, we discuss some related works about UP in the SAT domain before we indicate some paths for further research in the conclusion.

For space reasons, proofs of easy properties are omitted.

Preliminaries and notations

Let $\mathcal{L}$ be a standard Boolean logical language built on a finite set of Boolean variables (noted a, b, c, etc.) and usual Boolean connectives (namely, $\wedge$, $\vee$, $\neg$ and $\rightarrow$ standing for conjunction, disjunction, negation and material implication, respectively). Formulas will be noted using upper-case letters such as C, D, etc. Sets of formulas will be represented using Greek letters like Γ or Σ.

An interpretation is a truth assignment function that assigns values from $\{$*true*, *false*$\}$ to every Boolean variable, and thus, following usual compositional rules, to all formulas of $\mathcal{L}$. A formula is consistent or satisfiable when there exists at least one interpretation that satisfies it, i.e., that makes it become *true*.

Actually, any formula in $\mathcal{L}$ can be represented (while preserving satisfiability) in clausal normal form (CNF) using a set (interpreted conjunctively) of clauses, where a clause is a finite disjunction of literals and a literal is Boolean variable that can negated. We assume that a clause does not contain repeated occurrences of a literal. Clauses will be represented by the set of literals that they contain. For example, the clause $C = a \vee b \vee \neg c \vee \neg d$ will be represented by the set $\{a, b, \neg c, \neg d\}$. A CNF formula (in short, a CNF) will be represented by a set of clauses, i.e., a set of set of literals. A clause is said to be positive (resp. negative) when it contains no negative (resp. positive) literal. The size of a clause C, noted $|C|$, is the number of literals in C. Unit clauses contain exactly one literal whereas binary ones contain at most two literals. The empty clause is denoted by $\bot$ and is unsatisfiable. A clause C' is a sub-clause of C when $C' \subset C$ and C' is not empty. A sub-clause C' of C is called maximal when $|C| - |C'| = 1$. Note that the empty CNF, i.e., $\{\}$, is satisfiable whereas $\{\bot\}$ is unsatisfiable. For convenience, we will make no difference between $\{\bot\}$ and $\bot$.

SAT is the NP-complete problem that consists in checking

whether a CNF is satisfiable or not, i.e., whether there exists or not an interpretation that satisfies all clauses in the CNF.

A central deductive mechanism in this paper is the unit propagation mechanism (in short UP). UP is a linear time process that recursively simplifies a CNF by propagating the constraints expressed by unit clauses. We note Σ^* the CNF obtained by unit propagation on the CNF Σ. A clause C is a UP-consequence of Σ, noted $\Sigma \vdash^* C$, iff $(\Sigma \wedge \neg C)^*$ yields the empty clause. $\Sigma \models C$ indicates that C follows from Σ when C is satisfied in any interpretation satisfying Σ. Accordingly, when $\Sigma \vdash^* C$ then $\Sigma \models C$. The following concept of simplification will prove useful when discussing unit propagation: when Σ is a CNF and C is a clause then the simplification of Σ by C is defined by $\{D \setminus \neg C \mid D \in \Sigma \text{ and } D \cap C = \phi\}$ and noted Σ_C. Intuitively, Σ_C is obtained by simplifying Σ after substituting *true* for all literals in the clause C. We will abbreviate $(\Sigma_C)^*$ by Σ^*_C.

Some polynomial fragments of SAT

It is well-known that various clausal fragments of $\mathcal{L}$ exhibit polynomial-time algorithms for checking whether or not a CNF belongs to them and, in the positive case, for checking the satisfiability of the CNF. Among them, let us mention the Horn fragment (Dowling and Gallier 1984), which is made of Horn clauses only. A Horn (resp. reverse Horn) clause contains at most one positive (resp. negative) literal. Binary and renamable Horn clauses also form polynomial fragments: renamable Horn clauses are clauses that can be transformed into Horn by renaming i.e., by substituting a literal by its negation and vice-versa. Let us also mention Dalal's and Etherington's hierarchy of classes (Dalal and Etherington 1992) and the Q-Horn class (Boros, Crama, and Hammer 1990) (Boros, Hammer, and Sun 1994) formulas, which strictly contains all binary, reverse Horn and renamable Horn clauses. All of them can be recognized and solved in polynomial time; other variant polynomial fragments can be found in (Cepek and Kucera 2005) whereas various issues about Boolean forms are discussed in (Crama and Hammer 2011).

In this paper, we also consider one of the polynomial fragments of SAT introduced in (Tovey 1984): regardless of the number of variables per clause, when every variable occurs at most twice inside a CNF, this CNF can be solved in linear time.

Another polynomial fragment of $\mathcal{L}$ of interest in this paper is Quad (Dalal 1996). Quad is based on a tractable fragment called Root. A formula Σ is in class Root, if either (1) Σ contains the empty clause, or (2) Σ contains no positive clause, or (3) Σ contains no negative clause, or (4) all clauses of Σ are binary. A formula Σ is in class Quad if either (1) Σ^* belongs to Root, or (2) for the first maximum sub-clause C' of the first clause $C \in \Sigma^*$ for which $(\Sigma \wedge \neg C')^*$ is in class Root: (a) either $(\Sigma \wedge \neg C')^*$ is satisfiable, or (b) the formula $(\Sigma \setminus \{C\}) \cup \{C'\}$ is in class Quad. As emphasized by Dalal, Quad depends on the considered ordering of clauses. Different orderings can lead to different Quad classes.

In (Fourdrinoy et al. 2007), an extension of the Horn (resp. reverseHorn and binary) class[1] has been introduced that also uses UP as a cornerstone.

Definition 1. *Let $C = \{\neg n_1, \ldots, \neg n_r, p_1, \ldots, p_s\}$, with $r \geq 0$ and $s > 1$, be a clause of Σ. C is called a UP-Horn clause of Σ iff there exists a Horn clause $C' \subset C$ s.t. $\Sigma \vdash^* C'$.*

Then, the UP-Horn class is defined as the class of CNF that contain clauses that are Horn or UP-Horn. Clearly, checking whether a CNF belongs to UP-Horn and checking the satisfiability of a CNF belonging to this latter class can be achieved in polynomial time.

Based on similar considerations, UP-reverseHorn and UP-bin(ary) classes are easily defined.

Extensions of one Tovey's fragment

From now on, we call "Tovey's class" the class of CNF where any variable occurs at most twice (counting occurrences of both positive and negative literals).[2]

Definition 2. *Let Σ be a CNF, $\Sigma \in$ Tovey iff every variable appearing in Σ occurs at most twice in Σ.*

It is well-known that the (un)satisfiability of a CNF Σ is preserved when a clause C of Σ is replaced by one of its sub-clauses C', provided that $\Sigma \models C'$.

Property 1. *Let Σ be a CNF, C be a clause of Σ and C' be such that $C' \subseteq C$. When $\Sigma \models C'$ we have that Σ is satisfiable iff $(\Sigma \setminus \{C\}) \cup \{C'\}$ is satisfiable.*

The idea is to apply Property 1 to find polynomial-time recognizable and solvable extensions of Tovey; to this end, we will only apply the property when $\Sigma \vdash^* C'$ to benefit from the linear-time complexity of UP.

The class of CNF that can be reduced into Tovey's instances thanks to this property and by using unit propagation only, is defined as follows. We call it $\bigcup$UP-Tovey.

Definition 3. *Let Σ be a CNF, Σ belongs to $\bigcup$UP-Tovey iff $\exists \Gamma \in$ Tovey s.t. that there exists a one-to-one correspondence between Γ and Σ s.t. $\forall C \in \Sigma$, either $C \in \Gamma$ or $\exists C' \in \Gamma$ s.t. $C' \subset C$ and $\Sigma \vdash^* C'$.*

A straightforward algorithm checking whether a CNF belongs to $\bigcup$UP-Tovey or not is exponential time in the worst case. Hence, we look for subclasses of $\bigcup$UP-Tovey for which instances can be recognized in polynomial time.

A concept of a clause that is Tovey in a CNF will prove useful in the following.

Definition 4. *A clause C is a Tovey clause in Σ iff every variable occurring in C occurs at most twice in Σ.*

Note that this definition does not require a Tovey clause in Σ to belong to Σ. Obviously, a CNF Σ that belongs to Tovey is made of clauses that are Tovey in Σ. Let us stress that in Definition 3, C' is required to be Tovey in Γ. A very natural restriction of this constraint is to ensure that C' is Tovey in Σ. Obviously, since Γ is made of the clauses of Σ with some

[1] In (Fourdrinoy et al. 2007), the names of the resulting classes are actually noted using the U prefix. Here, for clarity of presentation and uniformity, we prefix their names by UP.

[2] Actually, Tovey defined other polynomial fragments, too.

Algorithm 1: Check-membership-in-UP-Tovey

Input: a CNF Σ made of m clauses
Output: a tentative reduction of Σ into a Tovey CNF ; *true* if $\Sigma \in$ UP-Tovey, *false* otherwise

1 $\Gamma \leftarrow \Sigma$;
2 **for** $(k = 1; k \leq m; k \leftarrow k + 1)$ **do**
3 $\quad C \leftarrow nextClause(\Sigma)$;
4 $\quad C' \leftarrow maxToveySubClause(C, \Sigma)$;
5 $\quad$ **if** $(C' \neq \emptyset \wedge \Sigma \vDash^* C')$ **then**
6 $\quad\quad \Gamma \leftarrow (\Gamma \backslash \{C\}) \cup \{C'\}$;
7 $\quad$ **else**
8 $\quad\quad$ **return** $(\Gamma$, *false*$)$;
9 **return** $(\Gamma$, *true*$)$;

of them having been shortened, a sub-clause C' that is Tovey in Σ is also Tovey in Γ.

We call UP-Tovey the resulting class of CNF.

Definition 5. *Let Σ be a CNF.*
A clause $C \in \Sigma$ is called a UP-Tovey clause in Σ iff $\exists C' \subseteq C$ s.t. C' is a Tovey clause in Σ and $\Sigma \vDash^* C'$.

Definition 6. *A CNF formula $\Sigma \in$ UP-Tovey iff every clause of Σ is a UP-Tovey clause in Σ.*

Clearly, we have that

Property 2. *UP-Tovey $\subset \bigcup$UP-Tovey*

Since UP is a linear-time process, checking the membership of a CNF in UP-Tovey and checking the satisfiability of a CNF belonging to UP-Tovey can both be achieved in polynomial time.

The following well-known property will prove useful to provide an algorithm that checks whether a given CNF belongs to UP-Tovey or not,

Property 3. *Let Σ be a CNF and C be a clause. If $\Sigma \nvDash^* C$ then $\forall C' \subset C$, $\Sigma \nvDash^* C'$.*

Accordingly, when a clause C is not UP-Tovey in Σ, it is sufficient to check whether the longest sub-clause C' that is Tovey in Σ is such that $\Sigma \vDash^* C'$ or not.

Definition 7. *Let Σ be a CNF and $C \in \Sigma$. A clause C' s.t. $C' \subset C$ is the maximum Tovey sub-clause of C in Σ iff C' is a Tovey clause in Σ and $\nexists C''$ s.t. that $C' \subset C'' \subset C$ and C'' is a Tovey clause in Σ.*

Property 4. *Let Σ be a CNF, $\Sigma \in$ UP-Tovey iff every clause C in Σ that is not Tovey in Σ contains a sub-clause C' that is a maximum Tovey sub-clause of C in Σ and such that $\Sigma \vDash^* C'$.*

A recognition algorithm for checking the membership of Σ in UP-Tovey needs thus to verify, for each clause $C \in \Sigma$ that is not Tovey in Σ, whether the maximum Tovey sub-clause of C in Σ can be deduced or not from Σ by unit propagation.

Algorithm 1 does the job. The resulting CNF is initialized to Σ. Clauses of Σ are processed one by one. The function $nextClause(\Sigma)$ selects the next clause C to process according to a static ordering on the clauses of Σ, but any such ordering can be used, this does not influence the result. C', the maximum Tovey sub-clause of C in Σ, is extracted. If such a clause is not empty and can be deduced from Σ by UP then the clause C is substituted in Γ by C'. In the other case, C is not UP-Tovey in Σ and the algorithm returns *false*.

In the worst-case, each clause is processed by unit propagation only once: the worst case time complexity of the algorithm is thus in $\mathcal{O}(m^2)$, where m is the number of clauses in the CNF.

Property 5. *Let Σ be a CNF of m clauses. The worst-case complexity of Algorithm 1 is in $\mathcal{O}(m^2)$.*

Algorithm 1 is complete for UP-Tovey.

Property 6. *Let Σ be a CNF, $\Sigma \in$ UP-Tovey iff $Check\text{-}membership\text{-}in\text{-}UP\text{-}Tovey(\Sigma)$ returns $true$.*

Interestingly, it is possible to improve Algorithm 1 in such a way that it attempts to recognize also (some) CNF that do not belong to UP-Tovey but that however belong to $\bigcup$UP-Tovey, while keeping a polynomial time worst-case complexity. It will allow the reduced instance Γ to contain clauses that are UP-Tovey in Γ but that are not UP-Tovey in Σ.

To this end, we take advantage of the following properties.

- Maximum Tovey sub-clauses can be checked in the currently reduced instance Γ under construction (instead of " in the initial Σ").
- There is no loss of relevant information by checking $\Gamma \vDash^* C'$ instead of checking $\Sigma \vDash^* C'$.
- Without altering the worst-case complexity, when we have $\Gamma \nvDash^* C'$, instead of ending the procedure, we can still process the other C clauses in Γ. The goal is to deliver a CNF instance with shortened clauses, so that the whole process can be iterated on this instance (and subsequent ones) with some possible additional shortenings being done at each iteration step.

Algorithm 3 implements these ideas. First, $Reduction$ (Algorithm 2) is called to tentatively shorten clauses of Σ. Then, it simply verifies whether the resulting CNF is Tovey ($isTovey(\Sigma)$ - line 2). Note that in $Reduction$ all instructions are achieved on Σ itself; thus Σ plays also the role

Algorithm 2: Reduction

Input: a CNF Σ made of m clauses
Output: a tentative reduction of Σ into a Tovey CNF

1 **for** $(k = 1; k \leq m; k \leftarrow k + 1)$ **do**
2 $\quad C \leftarrow nextClause(\Sigma)$;
3 $\quad C' \leftarrow maxToveySubClause(C, \Sigma)$;
4 $\quad$ **if** $(C' \neq \emptyset \wedge \Sigma \vDash^* C')$ **then**
5 $\quad\quad \Sigma \leftarrow (\Sigma \backslash \{C\}) \cup \{C'\}$;
6 **return** Σ;

Algorithm 3: Check-membership-in-$\bigcup$UP-Tovey

Input: a CNF Σ made of m clauses
Output: $true$ if a proof of $\Sigma \in \bigcup$UP-Tovey is found; $false$ otherwise

1 $\Sigma \leftarrow Reduction(\Sigma)$;
2 **return** $(isTovey(\Sigma))$;

Algorithm 4: Membership-to-G-UP-Tovey(i)

Input: a CNF Σ made of m clauses

Output: The lowest i s.t. $\Sigma \in$ UP-Tovey(i); 0 if such i does not exist

```
1 Σ_0 ← Σ;
2 j ← 1;
3 while (j ≤ m) do
4 |   Σ_j ← Reduction(Σ_{j−1});
5 |   if (isTovey(Σ_j)) then
6 |   |   return j;
7 |   else
8 |   |   j ← j + 1;
9 return 0;
```

of the CNF under construction. All clauses of Σ are processed one by one in the $Reduction$ procedure. The function $nextClause(\Sigma)$ (line 2) selects the next clause C to process according to a static ordering of the clauses of Σ. The procedure does not stop when one clause C that is not UP-Tovey in the current Σ has been discovered. It processes the remaining clauses in Σ. The hope is that C will actually become a Tovey clause in Σ at the end of the whole (iterated) reduction process. Obviously, Algorithm 3 remains incomplete for $\bigcup$UP-Tovey; especially, contrary to Algorithm 1, the order according to which clauses are processed (see $nextClause(\Sigma)$ in line 2 of $Reduction$) can influence the success of the reduction.

Moreover, when this algorithm returns *false*, some of the clauses of Σ might have been substituted by sub-clauses. Accordingly, Algorithm 3 can be iterated in order to recognize even more $\bigcup$UP-Tovey instances. In the sequel, we present a generalization of the UP-Tovey class by defining a hierarchy of classes, called G-UP-Tovey$_<(i)$ where $<$ is a given total ordering of clauses ("G" standing for"Generalized"). For convenience, we will drop the $<$ parameter indexing the class.

Definition 8. *Let Σ be a CNF made of m clauses. The sequence $\Sigma_0, \Sigma_1, \ldots, \Sigma_m$ is defined as follows.*

- $\Sigma_0 = Reduction(\Sigma)$
- $\Sigma_i = Reduction(\Sigma_{i-1})$, *where* $1 \leq i < m$.

$\forall i \in [0..(m-1)] : \Sigma \in$ *G-UP-Tovey*(i) *iff* Σ_i *is Tovey.*

Clearly, whenever $\Sigma_i = \Sigma_{i-1}$, no further change can occur in Σ_j where $j \geq (i-1)$. Since each call to $Reduction$ makes at least one clause become Tovey in the CNF under construction, it is guaranteed that applying $Reduction$ on Σ_m would not change Σ_m when m is the number of clauses in Σ.

Alternatively, we could have defined $\Sigma_i = (\Sigma_{i-1} \backslash C) \cup D$ for all $C \in \Sigma_{i-1}$ s.t. $\exists D \subset C$ a maximum Tovey sub-clause in Σ_{i-1} and $\Sigma_{i-1} \vDash^* D$.

Algorithm 4 depicts a way to determine for a given CNF Σ, whether there exists a lowest i such that Σ belongs to G-UP-Tovey(i) or not.

It is easy to show that G-UP-Tovey(i) $\subset$ G-UP-Tovey($i+1$) for $i \geq 1$. Indeed, any CNF formula Σ that belongs to G-UP-Tovey(i) also belongs to G-UP-Tovey(j) for $j > i$, whereas the converse is not true.

As the number of calls of Algorithm 4 to $Reduction$ is bounded by m, the overall complexity is then in $\mathcal{O}(m^3)$.

Property 7. *Let Σ be a CNF made of m clauses. The complexity of Algorithm 4 is in $\mathcal{O}(m^3)$.*

Clearly, we have:

Property 8. *Let $<$ be a total ordering of clauses. let n be any positive integer.*
UP-Tovey = G-UP-Tovey$(0) \subset$ *G-UP-Tovey*$(1) \subset ... \subset$ *G-UP-Tovey*$(n-1) \subset \bigcup$*UP-Tovey.*

Let us end this section by three remarks.

First, the G-UP-Tovey(i) tractable classes depend on the static ordering $<$ of the clauses. In the general case, to get rid of this dependence, we would need to consider all the possible orderings and the resulting recognition algorithm would then become exponential. A second remark is that even in the case where a static clauses ordering is considered, if we look for the smallest Tovey sub-clauses or simply the smallest sub-clause (line 3 of Algorithm 3) in the reduction phase, the recognition algorithm becomes exponential in the size of the longest clause. Finally, it must be noted that neither the UP-Tovey nor the G-UP-Tovey classes attempt to substitute a clause by a sub-clause that would contain variables occurring more than twice, although it might appear that other shortened clauses would decrease the total occurrences of these variables, making the reduced CNF become Tovey. This is another reason for the proposed classes to be strict sub-classes of $\bigcup$UP-Tovey.

UP-Horn (UP-reverseHorn, UP-bin) vs. Quad

Let us now turn our attention to UP-Horn (resp. UP-reverseHorn, UP-bin) and Quad. In (Fourdrinoy et al. 2007), it is shown that these two SAT fragments are incomparable due to the fact that Quad depends on a selected ordering of clauses while UP-Horn (resp. UP-reverseHorn, UP-bin) does not depend on any ordering of clauses. In this section, we answer a remaining question: are UP-Horn (resp. UP-reverseHorn, UP-bin) and Quad equivalent, provided that Quad is considered on all possible orderings of clauses? Let us note $\bigcup$Quad the union of all Quad classes obtained by considering all possible orderings of clauses. We show that UP-Horn (resp. UP-reverseHorn, UP-bin) is strictly included in $\bigcup$Quad.

Theorem 1.

A) UP-Horn $\subset \bigcup$*Quad*

B) UP-ReverseHorn $\subset \bigcup$*Quad*

C) UP-bin $\subset \bigcup$*Quad*

Proof. First of all, let us prove the following lemma.

Lemma 1. *Let Σ a CNF and C be a sub-clause of D s.t. $D \in \Sigma$ and $C \in \Sigma^*$.*

1. *If* $\Sigma \wedge \neg(D \setminus C) \vDash^* \bot$ *then* $\bot \in \Sigma^*$
2. *If* $\Sigma \wedge \neg(D \setminus C) \nvDash^* \bot$ *and* $F \subseteq D$ *is a clause s.t.* $\Sigma \wedge \neg F \vDash^* \bot$ *then* $\Sigma \wedge \neg(F \cap C) \vDash^* \bot$

Proof of Lemma 1.

1. Let $D = \{a_1, a_2, \ldots, a_n\}$ and $C = \{a_1, a_2, \ldots, a_r\}$ with $r \leqslant n$. $\Sigma \wedge \neg(D \setminus C) \models^* \bot$, can be written as follows:

$$\Sigma \wedge \bigwedge_{j \in \{r+1,\ldots,n\}} \{\neg a_j\} \models^* \bot \quad (1)$$

As C is obtained from D by unit propagation on Σ, this means that all literals a_j with $j \in \{r+1, \ldots, n\}$ are already propagated from Σ. In consequence, from (1) we have $\Sigma \models^* \bot$, in other words $\bot \in \Sigma^*$.

2. $\Sigma \wedge \neg(D \setminus C) \not\models^* \bot$ means that $\bot \notin \Sigma^*$ and for each sub-clause E of $(D - C)$, $\Sigma \wedge \neg E \not\models^* \bot$. If $F \subseteq D$ then $F \setminus (F \cap C) \subseteq (D - C)$, therefore

$$\Sigma \wedge \neg(F \setminus (F \cap C)) \not\models^* \bot \quad (2)$$

Assume that $\Sigma \wedge \neg(F \cap C) \not\models^* \bot$, with (2) we obtain that $\Sigma \wedge \neg F \not\models^* \bot$ which is in contradiction with the initial hypothesis.

□

Now, let us prove the first assertion, namely A).
Assume that Σ is UP-Horn and $\Sigma \notin \bigcup$Quad. We assume also that Σ^* contains positive clauses, because otherwise $\Sigma^* \in$ Root and thus $\Sigma \in \bigcup$Quad. Let P a positive clause of Σ^*. Let M a clause of Σ s.t. $P \subseteq M$.

- If $\Sigma \wedge \neg(M \setminus P) \models^* \bot$ then $\bot \in \Sigma^*$ (by Lemma 1.1) and hence $\Sigma \in \bigcup$Quad.
- If $\Sigma \wedge \neg(M \setminus P) \not\models^* \bot$ then $\forall E \subseteq (M \setminus P)$, $\Sigma \wedge \neg E \not\models^* \bot$. Σ is UP-Horn, so there exists a Horn clause H s.t. $H \subset M$ and $\Sigma \wedge \neg H \models^* \bot$. Therefore, $H \not\subseteq (M \setminus P)$ which means that $H \cap P \neq \emptyset$. H is a Horn clause and P is a positive one, in consequence $H \cap P = \{p\}$, i.e., $H \setminus \{p\} \subset (M \setminus P)$. By Lemma 1.2 we have: $\Sigma \wedge \{\neg p\} \models^* \bot$. So, for each positive clause P_i for $i = 1 \ldots n$, there is a variable p_i s.t.

$$\Sigma \wedge \{\neg p_i\} \models^* \bot \quad (3)$$

Now, let $C'_1, C'_2, \ldots, C'_{n'}$ be the clauses of Σ^* s.t. their negative literals are only formed by some of the $\neg p_i$'s. As previously, let D_i be a clause of Σ s.t. $C'_i \subseteq D_i$ ($i \in \{1, \ldots, n'\}$). Once again, we assume that $\Sigma \wedge \neg(D_i \setminus C'_i) \not\models^* \bot$ because Lemma 1.1 would direclty entail that Σ belongs to Root otherwise. So, we have:

$$\forall E' \subseteq (D_i \setminus C'_i), \Sigma \wedge \neg E' \not\models^* \bot \quad (4)$$

And we can apply the same reasoning: Σ is UP-Horn, thus $\exists H' \subset D_i$ s.t. H' is Horn and $\Sigma \wedge \neg H' \models^* \bot$ and since $H' \not\subseteq D_i \setminus C'_i$, we also have $H' \cap C'_i \neq \emptyset$. As H' is a Horn clause, two cases are to be distinguished (note that $(H' \setminus (H' \cap C'_i)) \subseteq D_i$ and consequently $(H' \setminus (H' \cap C'_i)) \subset D_i \setminus C'_i$):

1. $H' \cap C'_i = \{\neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\}$ has no positive literal. Since $\Sigma \wedge \neg H' \models^* \bot$ and (4), we have $\Sigma \wedge \neg\{\neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\} \models^* \bot$, i.e., $\Sigma \wedge (p_{i_1}) \wedge (p_{i_2}) \wedge \cdots \wedge (p_{i_m}) \models^* \bot$ which means that there exists a sub-clause $\{\neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\}$ of Σ s.t. $(\Sigma \wedge \neg\{\neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\})^*$ contains an empty clause and so belongs to Root which entails that $\Sigma \in$ Quad.
2. $H' \cap C'_i = \{p'_j, \neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\}$ contains one positive literal. Thanks to (3), (4) and $\Sigma \wedge \neg H' \models^* \bot$, we have:

$$\Sigma \wedge \{\neg p'_j\} \models^* \bot \quad (5)$$

Similarly, we can perform the same reasoning with $C''_1, C''_2, \ldots, C''_{n''}$ the clauses of Σ^* s.t. their negative literals are only formed by some of $\neg p_i$ and $\neg p'_j$, we obtain $\Sigma \wedge \{\neg p''_k\} \models^* \bot$, and so on.
Now, if we take the total order s.t. all p_i precede all p'_j which precede all p''_k and so on and all of these literals precede all the other ones of Σ (i.e., $p_1 < p_2 < \cdots < p_n < p'_1 < \cdots < p'_{n'} < p''_1 < \cdots < p''_{n''} < \cdots < p'^k_1 < \cdots < p'^k_{n'^k} < \ldots$), we can apply Dalal's procedure and it is possible to remove all p_i, all p'_j, all p''_k and so on. Thus the clauses that contain these positive literals are also removed. The same reasoning is applied until the resulting formula does not contain any positive clause; this latter formula belongs to Root, and consequently Σ is in Quad.

The second assertion, namely B), can be proved in a similar way.

The last assertion about UP-bin can be proved as follows.
Let Σ a CNF in UP-bin. For each non-binary clause $C = \{p_1, p_2, \ldots, p_n\}$ of Σ, we have $\Sigma \models^* \{p_i, p_j\}$ with $\{i, j\} \subseteq \{1, \ldots, n\}$ (because $\Sigma \in$ UP-bin) which means $\Sigma \models (\Sigma \setminus \{C\}) \cup \{p_i, p_j\}$, i.e., Σ can be replaced by $(\Sigma \setminus \{C\}) \cup \{p_i, p_j\}$. When this transformation is iterated for each non-binary clause, we obtain a binary formula, which is in Root, and thus Σ is in Quad.

To show that the inclusion is strict, consider the following CNF: $\Sigma = \{\{a, b, c\}, \{\neg d, \neg e, f\}\}$. It is easy to show that Σ belongs to Quad but does neither belong to UP-Horn, nor UP-ReverseHorn nor UP-bin.

First, Σ belongs to Quad. Indeed, since $\Sigma^* = \Sigma$ does not contain any negative clause, $\Sigma^* \in$ Root and thus $\Sigma \in$ Quad. Σ is not in UP-Horn because $(\Sigma \wedge \neg a)^* = \{\{b, c\}, \{\neg d, \neg e, f\}\} \not\models^* \bot$ and similar results hold for $(\Sigma \wedge \neg b)^*$ and $(\Sigma \wedge \neg c)^*$. Σ is not in UP-ReverseHorn because none of $\{\neg d\}, \{\neg e\}, \{f\}, \{\neg d, f\}$ and $\{\neg e, f\}$ follows from Σ through $\models^*$. Σ is not in U-bin. Indeed, no binary sub-clauses of $\{a, b, c\}$ and $\{\neg d, \neg e, f\}$ can be derived from Σ according to $\models^*$. □

To complete the comparison between the various fragments considered in this paper, let us end with the following results showing that $\bigcup$Quad and G-UP-Tovey are incomparable.

Theorem 2. *For all $i \geq 1$,*
$\bigcup$Quad $\not\subseteq$ G-UP-Tovey(i)
G-UP-Tovey(i) $\not\subseteq \bigcup$Quad

Proof. Let us build two counter-examples showing that $\subseteq$ does not hold. First, consider $\Sigma = \{\{a, \neg b\}, \{a, \neg c\}, \{a, \neg d\}\}$. Σ is in Quad since it does not contain any positive clause. However, Σ does neither belong to UP-Tovey, nor to UP-Tovey(i). Indeed, no sub-clause of

any clause of Σ can be deduced from Σ. Now, consider $\Psi = \{\{a, b, c\}, \{d, e, f\}, \{\neg a, \neg b, \neg c\}\}$. Clearly, Ψ belongs to Tovey and thus to G-UP-Tovey(i). However, Ψ does not belong to Root and no sub-clause can be derived according to any ordering of clauses. Consequently, Ψ is not in $\bigcup$Quad. □

Related work

Thanks to its linear-time cost, the unit propagation algorithm has been used in several ways in the context of SAT, in addition to being a key component of DPLL-like procedures. For example, C-SAT and Satz used a local treatment during important steps of the exploration of the search space, based on UP, to derive implied literals and detect local inconsistencies, and guide the selection of the next variable to be assigned (Dubois et al. 1996; Li and Anbulagan 1997). In (Le Berre 2001), a double UP schema is explored in the context of SAT solving. In (Ostrowski et al. 2003; Grégoire et al. 2004), UP has been used as an efficient tool to detect functional dependencies in SAT instances. The UP technique has also been exploited in (Darras et al. 2005) in order to derive sub-clauses by using the UP implication graph of the SAT instance, and speed up the resolution process. Unit propagation plays an important role on deciding the satisfiability of various well-known tractable formulas. Iterating the unit propagation technique is known to be complete on Horn CNF. Noticeably, a SLUR class (Franco and Gelder 2003; Vlcek et al. 2012) is formed of the CNF on which the single look-ahead unit resolution always succeeds. SLUR contains Horn, Hidden Horn (Lewis 1978), extended Horn (Chandru and Hooker 1991) and CC-Balanced formulas (Conforti, Cornuéjols, and Vuskovic 2006) as a subclass. However, checking whether a CNF belongs to SLUR is Co-NP-Complete (Cepek, Kucera, and Vlcek 2012), whereas in this paper the focus was on finding out tractable classes that can be recognized in polynomial time.

Conclusion

In this paper, the focus was on fragments of the clausal Boolean language for which instances are recognizable in polynomial time and satisfiability can be checked in polynomial time, too. First, a class called UP-Toveys has been defined whose instances can be recognized and solved in quadratric time. It extends one of the well-known Tovey's fragments. Also, we have shown that UP-Horn and other UP-based fragments are included in $\bigcup$Quad. Although this solves open questions about the relationships between these fragments, UP-Horn remains of high interest since it does not depend on any ordering of clauses whereas each such ordering gives rise to a specific Quad class.

Noticeably, although G-UP-Tovey(i) and $\bigcup$Quad are based on a same inference rule (i.e., entailment modulo unit propagation), their reduction processes are different. Accordingly, a promising path for further research would consist in investigating how these processes could be mixed and hybridized, giving rise to variant tractable classes.

Acknowledgments

Part of this work has been supported by the *Région Nord-Pas-de-Calais* and a FEDER grant. We are also grateful to the French ANR Agency and the TUPLES project for supporting this work. We thank the anonymous reviewers for their comments that helped us improve the paper.

References

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

Boros, E.; Hammer, P. L.; and Sun, X. 1994. Recognition of q-horn formulae in linear time. *Discrete Applied Mathematics* 55(1):1–13.

Cepek, O., and Kucera, P. 2005. Known and new classes of generalized Horn formulae with polynomial recognition and SAT testing. *Discrete Applied Mathematics* 149(1-3):14–52.

Cepek, O.; Kucera, P.; and Vlcek, V. 2012. Properties of slur formulae. In *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, 177–189. Springer.

Chandru, V., and Hooker, J. N. 1991. Extended Horn sets in propositional logic. *Journal of the ACM* 38(1):205–221.

Conforti, M.; Cornuéjols, G.; and Vuskovic, K. 2006. Balanced matrices. *Discrete Mathematics* 306(19–20):2411–2437.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151–158. New York (USA): Association for Computing Machinery.

Crama, Y., and Hammer, P. L. 2011. *Boolean Functions - Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of mathematics and its applications*. Cambridge University Press.

Dalal, M., and Etherington, D. W. 1992. A hierarchy of tractable satisfiability problems. *Information Processing Letters* 44(4):173–180.

Dalal, M. 1996. An almost quadratic class of satisfiability problems. In Wahlster, W., ed., *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96)*, 355–359. Budapest (Hungary): John Wiley & Sons, Ltd.

Darras, S.; Dequen, G.; Devendeville, L.; Mazure, B.; Ostrowski, R.; and Saïs, L. 2005. Using Boolean constraint propagation for sub-clauses deduction. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05)*, 757–761.

DIMACS. 1993. Second challenge on satisfiability testing.

Dowling, W. F., and Gallier, J. H. 1984. Linear-time algorithms for testing satisfiability of propositional Horn formulae. *Journal of Logic Programming* 267–284.

Dubois, O.; André, P.; Boufkhad, Y.; and Carlier, Y. 1996. *Second DIMACS implementation challenge: cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in*

Discrete Mathematics and Theoretical Computer Science. American Mathematical Society. chapter SAT vs. UNSAT, 415–436.

Fourdrinoy, O.; Grégoire, É.; Mazure, B.; and Saïs, L. 2007. Reducing hard SAT instances to polynomial ones. In *Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI 2007)*, 18–23.

Franco, J., and Gelder, A. V. 2003. A perspective on certain polynomial-time solvable classes of satisfiability. *Journal of Discrete Applied Mathematics* 125:177–214.

Grégoire, É.; Ostrowski, R.; Mazure, B.; and Saïs, L. 2004. Automatic extraction of functional dependencies. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, LNCS 3542, 122–132.

Le Berre, D. 2001. Exploiting the real power of unit propagation lookahead. In *Proceedings of the Fourth International Conference on Theory and Applications of Satisfiability Testing (SAT'01)*.

Lewis, H. R. 1978. Renaming a set of clauses as a Horn set. *Journal of the ACM* 25:134–135.

Li, C. M., and Anbulagan. 1997. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, 366–371.

Ostrowski, R.; Mazure, B.; Saïs, L.; and Grégoire, É. 2003. Eliminating redundancies in SAT search trees. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'2003)*, 100–104.

SAT-Competition. 2013. `http://www.satcompetition.org/`.

Tovey, C. A. 1984. A simplified NP-complete satisfiability problem. In *Discrete Applied Mathematics*, volume 8, 85–89.

Vlcek, V.; Balyo, T.; Gurský, S.; and Kucera, P. 2012. On hierarchies over the SLUR class. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2012), Fort Lauderdale, Florida, USA, January 9-11*.