Local autarkies searching for the dynamic partition of CNF formulae

Éric Grégoire, Bertrand Mazure, Lakhdar Saïs

Local autarkies searching for the dynamic partition of CNF formulae

Éric Grégoire Bertrand Mazure Lakhdar Saïs
Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens
{gregoire,mazure,sais}@cril.fr

Abstract

In this paper an original dynamic partition of formulae in Conjunctive Normal Form (CNF) is presented. It is based on the autarky concept first introduced by Monien and Speckenmeyer and further investigated by Kullmann and Van Gelder. Intuitively, an autarky is a partial assignment satisfying some clauses while not affecting any literal in any other clause, leading to a partition of the CNF formula. Autarkies can play a dramatic role in the efficiency of modern SAT solvers. The approach in this paper aims to dynamically extend the current partial assignment to a local autarky thanks to an inference rule based on unit propagation. More precisely, at each node of the search tree, it is checked whether the current decision literal can be made monotone by subsuming all the clauses where it appears negatively. The formal framework is detailed and its technical features discussed.

1. Introduction

The SAT problem, i.e., the problem of checking whether a set of Boolean clauses is satisfiable or not, is central in many computer science and artificial intelligence domains, including e.g. constraint satisfaction problems (CSP), planning, non-monotonic reasoning and VLSI correctness checking. Today, SAT has gained a considerable audience with the advent of a new generation of SAT solvers able to solve large complex instances encoding real-world applications. These solvers also appear to provide crucial base-building blocks for various other problem-solving technologies, like SMT solving, theorem proving, model finding and QBF solving.

These solvers, called modern SAT solvers [19, 6], are based on the standard unit propagation technique [4] combined in a smart way with efficient data structures, together with: (i) restart policies [7, 12], (ii) activity-based variable selection heuristics (VSIDS-like) [3, 19], and (iii) clause learning [17, 1, 19]. Modern SAT solvers can be interpreted as extended versions of the well-known DPLL-like (short for Davis, Putnam, Logemann and Loveland) procedure augmented with these various enhancements. Let us stress that the well-known resolution rule still plays a dramatic role in the efficiency of modern SAT solvers which can be understood as a particular form of general resolution [2]. These lookback-based SAT solvers are particularly efficient on SAT instances encoding practical applications. Other lookahead-based SAT solvers are designed using efficient local processing techniques and variable branching heuristics (e.g. Satz [14], kcnfs [5], March-DL [8]). This last family of solvers are particularly efficient on random and crafted SAT instances.

The recent dramatic breakthrough in solving complex industrial SAT instances can often be related to the presence of hidden structures exploited in a smart way by the SAT solvers. Indeed, such instances are very suitable for decomposition-based techniques. However, most of the decomposition techniques are much time consuming. Light divide-and-conquer approaches have been proposed previously, using for example structure-based variable ordering heuristics to recursively decompose the SAT instances [9]. In [15], a dynamic decomposition method based on hypergraph separators is proposed. The authors give a nice integration of the separator decomposition into variable ordering of a modern SAT solver that speedups the search on real-world SAT instances. Other approaches are based on the exploitation of autarky assignments originally proposed by Monien and Speckenmeyer [18] and further investigated by Kullmann [13] and Van Gelder [21]. More recently, Liffton and Sakallah [16] proposed a novel algorithm that searches for autarkies directly using a standard satisability solver and explore the potential of trimming autarkies in minimal unsatisable subsets (MUS) and minimal correction subsets (MCS) extraction flows. Let us stress that the dynamic exploitation of autarkies assignments in DPLL-like search procedures actually goes back to Oxusoff's and Rauzy's work [20]. In their paper, these latter authors pro-

posed several variable branching heuristics to dynamically partition the CNF formula into disjoint sets of clauses. More precisely, the proposed variable ordering heuristics aim to partition the formula: the next variable to assign is selected in such a way that its assignment leads to a new sub-formula included in previous ones at the same branch of the tree. The approach that will be presented in this paper follows this idea but is heuristic-independent. More precisely, let x be the next variable to assign, the proposed approach tries to subsume the clauses where $\neg x$ appears and then deduce that such a literal is monotone[1]. This clearly extends the local autarky assignments, using subclause deduction. Indeed, a partial assignment that is not a local autarky can be transformed into a local autarky.

The paper is organized as follows. After some preliminary definitions and notations, basic and local autarkies, Oxusoff's and Rauzy's dynamic approach are described in section 3. Then our generalized local autarkies are formally presented in section 4. Finally, we conclude by introducing several promising perspectives opened by this framework.

2. Technical background

2.1. Preliminary definitions and notations

A *CNF formula* Σ is a set (interpreted conjunctively) of *clauses*, where a clause is a disjunction of *literals*. A literal is a positive (x) or negated ($\neg x$) propositional variable. The two literals x and $\neg x$ are called *complementary*. We note by $\bar{l}$ the complementary literal of l. For a set of literals L, $\bar{L}$ is defined as $\{\bar{l} \mid l \in L\}$. A *unit* clause is a clause containing only one literal (called *unit literal*), while a binary clause contains exactly two literals. The *empty clause*, noted $\bot$, is interpreted as $false$ (unsatisfiable), whereas the *empty CNF formula*, noted $\top$, is interpreted as $true$ (satisfiable). We define the size $|\Sigma|$ of a CNF formula Σ as $\sum_{c\in\Sigma} |c|$, where $|c|$ is the number of literals in c. The number of clauses in Σ is denoted C_Σ.

The set of variables occurring in Σ is noted V_Σ. A set of literals is *complete* if it contains one literal for each variable in V_Σ, and *fundamental* if it does not contain complementary literals. An *assignment* ρ of a Boolean formula Σ is a function that associates a value $\rho(x) \in \{false, true\}$ to some of the variables $x \in V_\Sigma$. ρ is *complete* if it assigns a value to every $x \in V_\Sigma$, and *partial* otherwise. An assignment is alternatively represented by a fundamental set of literals, in the obvious way. A *model* of a formula Σ is an assignment ρ that makes the formula $true$; noted $\rho \vDash \Sigma$.

The following notations will also be used throughout the paper:

- $\Sigma|_x$ will denote the formula obtained from Σ by assigning x the truth-value *true*. Formally $\Sigma|_x = \{c \mid c \in \Sigma, \{x, \neg x\} \cap c = \emptyset\} \cup \{c \backslash \{\neg x\} \mid c \in \Sigma, \neg x \in c\}$ (that is: the clauses containing x are removed; and those containing $\neg x$ are simplified). This notation is extended to assignments: given an assignment $\rho = \{x_1, \ldots, x_n\}$, $\Sigma|_\rho$ is defined as $(\ldots((\Sigma|_{x_1})|_{x_2})\ldots|_{x_n})$.
- Σ^* denotes the formula Σ closed under unit propagation, defined recursively as follows: (1) $\Sigma^* = \Sigma$ if Σ does not contain any unit clause, (2) $\Sigma^* = \bot$ if Σ contains two unit-clauses $\{x\}$ and $\{\neg x\}$, (3) otherwise, $\Sigma^* = (\Sigma|_x)^*$ where x is the literal appearing in a unit clause of Σ. A clause c is deduced by unit propagation from Σ, noted $\Sigma \vDash^* c$, if $(\Sigma|_{\bar{c}})^* = \bot$.

Let c_1 and c_2 be two clauses of a formula Σ. We say that c_1 (respectively c_2) subsumes (respectively is subsumed by) c_2 (respectively by c_1) iff $c_1 \subseteq c_2$. If c_1 subsumes c_2, then $c_1 \vDash c_2$ (the converse is not true). Also Σ and $\Sigma - c_2$ are equivalent with respect to satisfiability.

2.2. DPLL search

DPLL [4] is a tree-based backtrack search procedure; at each node of the search tree, the assigned literals (both the decision literal and the propagated ones) are labeled with the same *decision level* starting from 1 and increased at each decision (or branching). After backtracking, some variables are unassigned, and the current decision level is decreased accordingly. At the ith level, the current partial assignment ρ can be represented as a sequence of decision-propagation steps of the form $\langle (x^i_k), x^i_{k_1}, x^i_{k_2}, \ldots, x^i_{k_{n_k}} \rangle$ where the first literal x^i_k corresponds to the decision literal x_k assigned at the ith level and each $x^i_{k_j}$ for $1 \leq j \leq n_k$ represents unit propagated literals at the ith level. Let $x \in \rho$, we note $\delta(x)$ the assignment level of x.

3. Autarky assignments

The autarky concept has been introduced in [18].

Definition 1 *Let Σ be a CNF formula and ρ a partial assignment. ρ is an autarky if $\Sigma|_\rho \subseteq \Sigma$ i.e. $\forall c \in \Sigma$, $\bar{\rho} \cap c = \emptyset$ or $\rho \cap c \neq \emptyset$.*

For a CNF formula, computing an autarky when it exists is clearly (F)NP-complete. However, autarkies can be used to improve the efficiency of SAT solvers [21, 13]. Indeed, the refutation of $\Sigma|_\rho$ leads to the proof of the unsatisfiability of Σ. Following definition 1, the property prop:inclusion can be obtained.

Property 1 *If ρ is an autarky of the CNF formula Σ then $\Sigma \vDash \Sigma|_\rho$.*

1 A monotone literal is a literal that occurs either positively or negatively in the CNF formula (but not in both polarities).

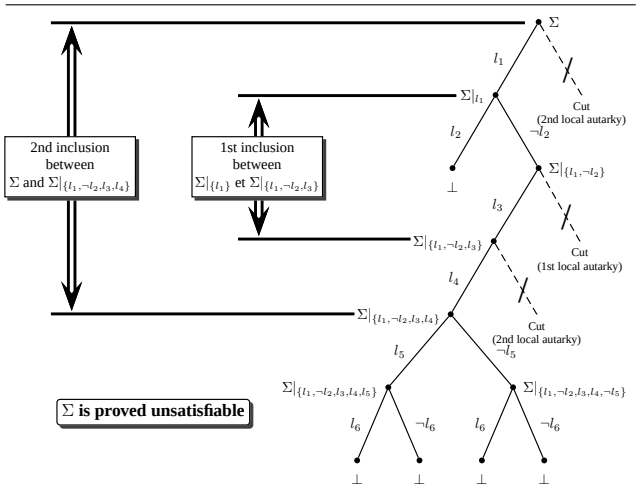

Figure 1. DPLL search tree and local autarkies

In [10, 20] the authors exploit this property to prune the search tree of DPLL-like procedures. To this end, a local autarky concept is defined and exploited.

Definition 2 *A partial assignment $\rho = \{l_0, \ldots, l_i\}$ of a CNF formula Σ is a local autarky if $\exists \rho' = \{l_0, \ldots, l_j\}$ with $j < i$ such that $\Sigma|_\rho \subseteq \Sigma|_{\rho'}$.*

Property 1 also applies for local autarkies. Indeed, from Definition 2 and Property 1, $\Sigma|_{\rho'} \models \Sigma|_\rho$ can be deduced. This can be used to prune the search tree of the DPLL algorithm. Indeed, if ρ is a local autarky with respect to the partial assignment ρ' then $\Sigma|_{\rho'}$ is satisfiable if and only if $\Sigma|_\rho$ is satisfiable. All branches not explored between ρ' and ρ can be pruned if Σ_ρ is proved unsatisfiable.

The following example illustrates this property.

Example 1 *Let* $\Sigma = \{(l_1 \vee l_2 \vee \neg l_4), (\neg l_1 \vee l_4), (\neg l_1 \vee l_2 \vee l_3), (\neg l_2), (l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\}$.

The search tree of the DPLL algorithm represented in Figure 1 is obtained thanks to the exploitation of local autarkies. This corresponds to the application of the local autarky pruning rule as defined by Jeannicot et al. in [10]. In the following example, the unit and monotone literals propagated during search are not mentioned for clarity reasons.

From the search tree, the following formulae associated to the different nodes are obtained:

- $\Sigma|_{\{l_1\}} = \{(l_4), (l_2 \vee l_3), (\neg l_2), (l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\}$
- $\Sigma|_{\{l_1,\neg l_2\}} = \{(l_4), (l_3), (l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\}$
- $\Sigma|_{\{l_1,\neg l_2,l_3\}} = \{(l_4), (l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\}$
- $\Sigma|_{\{l_1,\neg l_2,l_3,l_4\}} = \{(l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\}$
- $\Sigma|_{\{l_1,\neg l_2,l_3,l_4,l_5\}} = \{(l_6), (\neg l_6)\}$
- $\Sigma|_{\{l_1,\neg l_2,l_3,l_4,\neg l_5\}} = \{(l_6), (\neg l_6)\}$

Consequently, the following inclusion holds:

$$\left.\begin{array}{lcl} \Sigma|_{\{l_1,\neg l_2,l_3\}} & \subset & \Sigma|_{\{l_1,\neg l_2\}} \\ \Sigma|_{\{l_1,\neg l_2,l_3\}} & \subset & \Sigma|_{\{l_1\}} \end{array}\right) \text{First application of pruning rules of local autarky}$$

$$\left.\begin{array}{lcl} \Sigma|_{\{l_1,\neg l_2,l_3,l_4\}} & \subset & \Sigma|_{\{l_1,\neg l_2,l_3\}} \\ \Sigma|_{\{l_1,\neg l_2,l_3,l_4\}} & \subset & \Sigma|_{\{l_1,\neg l_2\}} \\ \Sigma|_{\{l_1,\neg l_2,l_3,l_4\}} & \subset & \Sigma|_{\{l_1\}} \\ \Sigma|_{\{l_1,\neg l_2,l_3,l_4\}} & \subset & \Sigma \end{array}\right) \text{Second application of pruning rules of local autarky}$$

Clearly, the local autarky pruning rule is a generalization of the monotone literal pruning one. Indeed, if l is a monotone literal, then $\Sigma|_l$ is the formula obtained from Σ where all clauses containing l are removed. In consequence $\Sigma|_l \subset \Sigma$ and $\{l\}$ is a local autarky for Σ. Using the pruning rule based on local autarky allows the simplification by means of the monotone literal rule. Let us note that monotone literals are not exploited in modern SAT solvers that exploit lazy data structures like watched literals. In other words, the occurrence number of a given literal at each node of the search tree is not recorded or updated.

At a given level i of the search tree, finding the smallest level $j < i$ such that $\Sigma|_{\{l_0,\ldots,l_i\}}$ is included in $\Sigma|_{\{x_0,x_1,\ldots,x_j\}}$ is computationally hard. However the following property allows an incremental detection that ensures the best pruning.

Property 2 ([20]) *If $\Sigma|_{\{l_0,\ldots,l_i\}} \subseteq \Sigma|_{\{l_0,\ldots,l_j\}}$ with $j < i$, then $\Sigma|_{\{l_0,\ldots,l_i\}}$ is also included in $\Sigma|_{\{l_0,\ldots,x_{j+1}\}}$, $\Sigma|_{\{l_0,\ldots,l_{j+2}\}}, \ldots,$ and $\Sigma|_{\{l_0,\ldots,l_{i-1}\}}$.*

To avoid useless inclusion tests, it is more convenient to use the following corollary.

Corollary 1 *If $\Sigma|_{\{l_0,\ldots,l_i\}} \not\subset \Sigma|_{\{l_0,\ldots,l_j\}}$, then $\Sigma|_{\{l_0,\ldots,l_i\}}$ is not included in all the following formulas: $\Sigma|_{\{l_0,\ldots,l_{j-1}\}}$, $\Sigma|_{\{l_0,\ldots,l_{j-2}\}}, \ldots, \Sigma|_{\{l_0\}}$ and Σ.*

The efficiency of the detection of the greatest value $i-j$, called a partition, depends only on the efficiency of the inclusion test between two formulas associated to two consecutive nodes in the current branch of the search tree. This test can be performed efficiently using adapted data structures.

Finally, to ensure that all inclusions are detected, it is necessary to remove subsumed clauses during the DPLL procedure, as illustrated by the following example.

Example 2 *Let* $\Sigma = \{(\neg l_1 \vee l_3), (l_2 \vee l_4), (\neg l_2 \vee l_3 \vee l_4)\}$, *and* $\Sigma|_{\{l_1\}} = \{(l_3), (l_2 \vee l_4), (\neg l_2 \vee l_3 \vee l_4)\}$ $\Sigma|_{\{l_1,l_2\}} = \{(l_3), (l_3 \vee l_4)\}$

In this case no inclusion is detected. But if the subsumed clauses are removed from $\Sigma|_{\{l_1,l_2\}}$*, then* $\Sigma|_{\{l_1,l_2\}} = \{(l_3)\} \subset \Sigma|_{\{l_1\}}$.

The subsumption test can be achieved efficiently using for example the recent approach by Lintao Zhang [22].

In the next section, a generalization of the (local) autarky assignments is introduced.

4. Generalized (local) autarky

The generalization is obtained by substituting the syntactical inclusion operation ($\subset$) by the semantic logical consequence relationship ($\models$) in Definitions 1 and 2.

Proposition 1 (Generalized (local) autarky) *Let* Σ *be a CNF formula,* $\rho = \{l_0, \ldots, l_j, \ldots, l_i\}$ *is a generalized local autarky if there exists* $\rho' = \{l_0, \ldots, l_j\}$ *such that* $\Sigma|_{\rho'} \models \Sigma|_{\rho}$*. If* $\rho' = \emptyset$ *then* ρ *is a generalized autarky.*

This is clearly a generalization of the previous concept, because if $\Omega \subset \Sigma$ then $\Sigma \models \Omega$ while the converse is false, as illustrated by the following example.

Example 3 *Let* $\Sigma = \{(a \vee b \vee c), (\neg a \vee b)\}$ *and* $\Omega = \Sigma|_{\neg a} = \{(b \vee c)\}$*, clearly* $\Sigma \models \Omega$ *but* $\Omega \not\subset \Sigma$.

The most important feature of local autarkies is the incremental detection which ensures the greatest inclusion or partition (see Property 2). This property is preserved for the generalized autarky assignments.

Property 3 *If* $\Sigma|_{\{l_0,\ldots,l_j\}} \models \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$*, then* $\Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$ *is a logical consequence of all the following formulae:* $\Sigma|_{\{l_0,\ldots,l_{j+1}\}}$, $\Sigma|_{\{l_0,\ldots,l_{j+2}\}}$, $\ldots$ *and* $\Sigma|_{\{l_0,\ldots,l_{i-1}\}}$.

As before, the following corollary avoids useless logical consequence tests (CoNP-Complete) in order to compute the greatest partition.

Corollary 2 *If* $\Sigma|_{\{l_0,\ldots,l_j\}} \not\models \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$*, then* $\Sigma|_{\{l_0,\ldots,l_i\}}$ *is not a logical consequence of the following formulae:* $\Sigma|_{\{l_0,\ldots,l_{j-1}\}}$, $\Sigma|_{\{l_0,\ldots,l_{j-2}\}}$, $\ldots$, $\Sigma|_{\{l_0\}}$ *and* Σ.

We have seen that the propagation of monotone literals is a special case of the local autarky detection. The following property shows that the classical unit propagation is also a special case of the generalized local autarkies.

Property 4 *Let* Σ *be a CNF formula and* l *a literal occurring in a unit clause of* Σ*, we have* $\Sigma \models \Sigma|_{\{l\}}$.

By extension, it can be shown that:

Corollary 3 *Let* Σ *a CNF formula,* $\Sigma \models \Sigma^*$.

More generally, if $\Sigma \models l$ with l a literal of the CNF formula Σ, then $\Sigma \models \Sigma|_{\{l\}}$. This means that the detection of generalized local autarkies includes the propagation due to any implied literal.

The use of generalized local autarkies within a DPLL algorithm can be achieved in the same way than for local autarkies. Unfortunately, if the test of inclusion can be computed efficiently for local autarkies, the detection of generalized autarkies requests a CoNP-complete test.

4.1. Generalized (local) autarky modulo unit propagation

In order to pratically exploit the generalized local autarkies, it is necessary to weaken the deductive relation that is used. Accordingly, the full consequence relationship ($\models$) is weakened down to the mere unit propagation rule ($\models^*$), which can be computed in linear-time.

Definition 3 *Let* Σ *and* Ω *be two sets of clauses,* Ω *is a deductive consequence of* Σ *restricted to unit propagation, noted* $\Sigma \models^* \Omega$*, if and only if* $\forall c \in \Omega$, $\Sigma \models^* c$.

Clearly, if $\Sigma \models^* \Omega$ then $\Sigma \models \Omega$. In consequence, Proposition 1 can be adapted to introduce the generalized local autarky modulo unit propagation.

Proposition 2 (Generalized (local) autarky modulo unit propagation) *Let* Σ *be a CNF formula,* $\rho = \{l_0, \ldots, l_j, \ldots, l_i\}$ *is a generalized local autarky modulo unit propagation if there exists* $\rho' = \{l_0, \ldots, l_j\}$ *such that* $\Sigma|_{\rho'} \models^* \Sigma|_{\rho}$*. If* $\rho' = \emptyset$ *then* ρ *is a generalized autarky modulo unit propagation.*

Earlier in the paper, it has been shown that the propagations of monotone, unit and more generally implied literals are special cases of local autarky detection. Likewise, the detection of local autarky modulo unit propagation automatically computes the propagations of monotone and unit literals, too. The following property proves useful in that respect:

Property 5 *Let* Σ *and* Ω *be two CNF formulae. If* $\Omega \subseteq \Sigma$*, then* $\Sigma \models^* \Omega$.

This property is easily proved. Indeed, each clause c of Ω is present in Σ, also $(\Sigma|_{\bar{c}})^* = \bot$ because the clause c of Σ is reduced to an empty clause by the propagation of $\bar{c}$. In consequence, for each clause c of Ω $\Sigma \models^* c$ and by definition $\Sigma \models^* \Omega$. The converse is false, as illustrated by the following example.

Example 4 *Let* $\Sigma = \{(a \vee b \vee c), (a \vee \neg b), (b \vee \neg c), (c \vee \neg a)\}$ *and* $\Omega = \Sigma|_b = \{(a), (c \vee \neg a)\}$*, we have* $\Sigma \models^* \Omega$ *but* $\Omega \not\subset \Sigma$.

So, Property 5 ensures that all classical local autarkies are also detected by generalized local autarkies modulo unit propagation. In consequence, the simplification of monotone literals is implicitly included during the detection of generalized local autarkies modulo unit propagation. Similarly, the simplification by unit propagation is a special case of generalized local autarkies modulo unit propagations (Property 6).

Property 6 *For any set of clauses* Σ, $\Sigma \vDash^* \Sigma^*$.

Compared with generalized autarkies, only the simplification by the implied literals are not included in the generalized autarkies modulo unit propagation, as illustrated by the following example:

Example 5 *Let* $\Sigma = \{(a \vee b \vee c), (a \vee b \vee \neg c), (a \vee \neg b \vee \neg c), (\neg a \vee b \vee \neg c), (\neg a \vee \neg b \vee c), (a \vee \neg b \vee c)\}$. *We have* $\Sigma \vDash a$ *but* $\Sigma \nvDash^* a$.

However, the restriction has been introduced to ensure that generalized (local) autarkies modulo unit propagation can be computed in polynomial time. Its implementation in a DPLL-like procedure is discussed in the next section.

4.2. Some useful properties and implementation

In this section, some practical properties about the generalized (local) autarkies modulo unit propagation are presented. More precisely, it is shown how the detection of these kinds of autarkies can be computed efficiently during the computation of a DPLL-like algorithm.

The first property is about the time complexity of deduction restricted to unit propagation.

Property 7 *Let* Σ *and* Ω *be two sets of clauses. Checking* $\Sigma \vDash^* \Omega$ *is in* $\mathcal{O}(C_\Omega \times (|\Sigma| + |c|))$ *where* c *is the longest clause of* Ω.

Unfortunately a polynomial-time complexity does not guarantee practical efficiency. In this respect, we also reduce in a significant way the number of unit propagations that are needed to detect local autarkies modulo unit propagation.

Property 8 *Let* Σ *a set of clauses and* l *a literal.* $\Sigma \vDash^* \Sigma|_l$ *if and only if* $\forall c \in \Sigma$ *such that* $\bar{l} \in c$, $\Sigma \vDash^* c$.

The above property suggests that after the assignment of a literal l, clauses containing $\neg l$ need to be checked, only.

From properties 7 and 8, the following corollary is easily obtained:

Corollary 4 *Let* Σ *be a set of clauses and* l *a literal. Checking* $\Sigma \vDash^* \Sigma|_l$ *is in* $\mathcal{O}(Occ(\bar{l}) \times C_\Sigma)$ *where* $Occ(\bar{l})$ *is the number of occurrences of* $\bar{l}$ *in* Σ.

Algorithm 1: `DPLL-autark`

input : A CNF formula Σ
output: true if Σ is satisfiable, false otherwise

1.1 $\Sigma^* \leftarrow$ `UnitPropagation(`Σ`)`;
1.2 **if** $\Sigma^* = \bot$ **then return** false;
1.3 **else if** $\Sigma^* = \emptyset$ **then return** true;
1.4 **else**
1.5 $\quad l \leftarrow$ `DecisionHeuristic(`Σ^*`)`;
1.6 $\quad$ **if** `DPLL-autark(`$\Sigma^* \wedge (l)$`)` **then return** true;
1.7 $\quad$ **else return** `AutarkyModuloUP(`Σ^*,l`)`;
1.8 **end**

This corollary shows that detecting a generalized local autarky modulo unit propagation between two consecutive nodes of a DPLL-like algorithm is a simple task that is very cheap w.r.t. time-complexity. However, there can exist generalized local autarkies modulo unit propagation between two non-consecutive nodes. The following property and its corollary show how useless tests can be avoided and that an incremental detection is possible.

Property 9 *Let* Σ *be a set of clauses and* $\{l_0, \ldots, l_j, \ldots, l_i\}$ *a set of literals. If* $\Sigma|_{\{l_0,\ldots,l_j\}} \vDash^* \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$, *then* $\Sigma|_{\{l_0,\ldots,l_j,l_{j+1}\}} \vDash^* \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$, $\Sigma|_{\{l_0,\ldots,l_j,l_{j+1},l_{j+2}\}} \vDash^* \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$, $\ldots$ *and* $\Sigma|_{\{l_0,\ldots,l_j,\ldots,l_{i-1}\}} \vDash^* \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$.

Corollary 5 *Let* Σ *be a set of clauses and* $\{l_0, \ldots, l_j, \ldots, l_i\}$ *a set of literals. If* $\Sigma|_{\{l_0,\ldots,l_j\}} \nvDash^* \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$, *then* $\Sigma|_{\{l_0,\ldots,l_{j-1}\}} \nvDash^* \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$, $\Sigma|_{\{l_0,\ldots,l_{j-2}\}} \nvDash^* \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$, $\ldots$ *and* $\Sigma \nvDash^* \Sigma|_{\{l_0,\ldots,l_j,\ldots,l_i\}}$.

Detecting generalized local autarkies modulo unit propagation between two non-consecutive nodes of a DPLL-like algorithm depends on such a detection between two consecutive nodes. Accordingly, we insert this detection process within a DPLL-like procedure. It is performed during the backtrack step. Compared with a forward detection, this method avoids the useless unit propagations when the developed branch is proved unsatisfiable. The following algorithms describe this detection process (see Algorithm 2) and its integration within a DPLL-like procedure (see Algorithm 1).

Algorithm 1 details the `DPLL-autark` function which is a DPLL-like algorithm using the simplification by the generalized (local) autarkies modulo unit propagation. In this algorithm, the `UnitPropagation` and `DecisionHeuristic` functions are the DPLL-like solvers usual ones. All kinds of decision heuristics (MOMS [11], VSIDS [19], etc.) can be used within `DPLL-autark`. When a call to the `AutarkyModuloUP` function is made, the $\Sigma|_l$ branch has been already explored and proved unsatisfiable. If $\Sigma \vDash^* \Sigma|_l$ then the $\Sigma|_{\bar{l}}$ branch can be pruned since

Algorithm 2: `AutarkyModuloUP`

input : a set of clauses Σ and a literal l such that $\Sigma|_l$ has been proved unsatisfiable
output: true if $\Sigma|_{\bar{l}}$ is satisfiable, false otherwise

2.1 $(\Sigma|_{\bar{l}})^* \leftarrow$`UnitPropagation(`$\Sigma \wedge (\bar{l})$`)`;
2.2 **if** $(\Sigma|_{\bar{l}})^* = \bot$ **then return** false;
2.3 **else if** $(\Sigma|_{\bar{l}})^* = \emptyset$ **then return** true;
2.4 **else**
2.5 autarkyFound $\leftarrow$true;
// $\Sigma \models^* \Sigma|_l$ *iff* $\forall c = (\bar{l} \vee c') \in \Sigma$, $\Sigma \models^* c'$
2.6 **foreach** $c \in \Sigma$ *s.t.* $c = (\bar{l} \vee l_1 \vee l_2 \vee \cdots \vee l_n)$ **do**
2.7 **if** `UnitPropagation(`$(\Sigma|_{\bar{l}})^* \wedge (\bar{l_1}) \wedge (\bar{l_2}) \wedge \cdots \wedge (\bar{l_n})$`)`$= \bot$ **then**
2.8 $(\Sigma|_{\bar{l}})^* \leftarrow (\Sigma|_{\bar{l}})^* \wedge (l_1 \vee l_2 \vee \cdots \vee l_n)$;
2.9 **end**
2.10 **else** autarkyFound $\leftarrow$false;
2.11 **end**
2.12 **if** autarkyFound **then return** true;
2.13 **else return** `DPLL-autark(`$(\Sigma|_{\bar{l}})^*$`)`;
2.14 **end**

an autarky is detected. The `AutrakyModuloUP` function computes this test and, depending on the result, it either explores or prune the branch. This function is detailed in Algorithm 2.

For efficiency reasons, Algorithm 2 begins with the unit propagation of $\bar{l}$. Indeed, each clause tested during the verification of $\Sigma \models^* \Sigma_l$ implies the unit literal $\bar{l}$. In order to avoid several propagations of $\bar{l}$, the propagation of $\bar{l}$ is done once at the beginning of the procedure (see line 2.1).

When a clause $c' = (l_1 \vee l_2 \vee \cdots \vee l_n)$ is proved to be a logical consequence through unit propagation of $(\Sigma|_{\bar{l}})^*$, it is added to $(\Sigma|_{\bar{l}})^*$ (see line 2.8). These additional clauses speed up the computation of all unit propagations done during the rest of the procedure and reduce the size of the subtree built during the exploration of the branch $\Sigma|_{\bar{l}}$ when no autarky has been detected. Note that these additions are performed in constant space. Indeed, the c' clause is obtained from a c clause of Σ that contains $\bar{l}$. This c clause is satisfied during the unit propagation of $\bar{l}$ at the beginning of the procedure. The addition of c' can be done with a simple removal of the $\bar{l}$ literal from c. This process can be achieved in constant time and space with an adequate data structure.

Finally, even if the generalized local autarky modulo unit propagation is not detected, this approach is able to produce several sub-clauses. It is also possible to stop the detection as soon as $\Sigma \not\models^* \Sigma|_l$ is proved (at the line 2.10). We believe that the spent time to produce sub-clauses is offset by the time saved to explore a smaller sub-tree. Two versions can be derived, one that only checks for generalized local autarkies modulo unit propagation and an other one that additionally produces sub-clauses. In the future, we plan to implement these two versions in order to measure and compare their practical strengths.

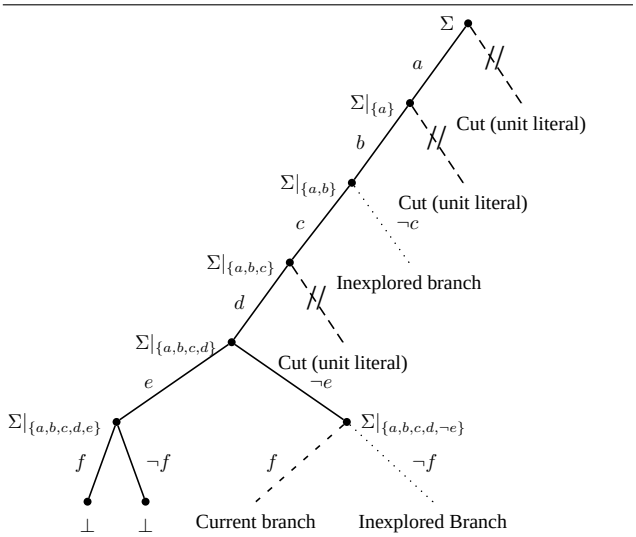

Figure 2. Levels of literals in DPLL

CNF Formula	Levels of literals											
	$\delta(a)$	$\delta(\bar{a})$	$\delta(b)$	$\delta(\bar{b})$	$\delta(c)$	$\delta(\bar{c})$	$\delta(d)$	$\delta(\bar{d})$	$\delta(e)$	$\delta(\bar{e})$	$\delta(f)$	$\delta(\bar{f})$
Σ	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\Sigma\|_{\{a\}}$	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\Sigma\|_{\{a,b\}}$	0	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\Sigma\|_{\{a,b,c\}}$	0	∞	0	∞	1	∞	∞	∞	∞	∞	∞	∞
$\Sigma\|_{\{a,b,\bar{c}\}}$	0	∞	0	∞	∞	0	∞	∞	∞	∞	∞	∞
$\Sigma\|_{\{a,b,c,d\}}$	0	∞	0	∞	1	∞	1	∞	∞	∞	∞	∞
$\Sigma\|_{\{a,b,c,d,e\}}$	0	∞	0	∞	1	∞	1	∞	2	∞	∞	∞
$\Sigma\|_{\{a,b,c,d,e,f\}}$	0	∞	0	∞	1	∞	1	∞	2	∞	3	∞
$\Sigma\|_{\{a,b,c,d,e,\bar{f}\}}$	0	∞	0	∞	1	∞	1	∞	2	∞	∞	2
$\Sigma\|_{\{a,b,c,d,\bar{e}\}}$	0	∞	0	∞	1	∞	1	∞	∞	1	∞	∞
$\Sigma\|_{\{a,b,c,d,\bar{e},f\}}$	0	∞	0	∞	1	∞	1	∞	∞	1	2	∞
$\Sigma\|_{\{a,b,c,d,\bar{e},\bar{f}\}}$	0	∞	0	∞	1	∞	1	∞	∞	1	∞	1

Table 1. Levels of literals of the search tree depicted in Fig. 2.

4.3. Local autarkies and implication levels

In the previous algorithm, only the current branch of DPLL can be cut when an autarky is found. It could be interesting to cut several branches to process a non-chronological backtracking like it is done in modern implementations of DPLL. In this section, it is shown that this goal can be reached by computing the largest jump with respect to an order of unit propagations. The standard concepts of level for a literal (section 2.2) and implication graph (Definition 5) are used in that respect.

We recall that $\delta(l)$ the level of a given literal l is given by the number of branches not explored before its assignment.

All assigned literals between two calls of the decision heuristics have the same level, as illustrated in the next example. An unassigned literal has an infinite level. In Table 1, the level of all literals at each node of the search tree depicted in Figure 2 is detailed.

Definition 4 *Let Σ be a CNF formula, $\{l_1, l_2, \ldots, l_i\}$ a partial assignment of Σ and c a clause of $\Sigma|_{\{l_1,l_2,\ldots,l_i\}}$. The level of c, noted $\delta(c)$, is defined in the following way:*

- *If $c \in \Sigma$, $\delta(c) = 0$;*
- *Else $\delta(c) = \max\{\delta(l)$ s.t. $l \in c$ et $\bar{l} \in \{l_1, l_2, \ldots, l_i\}\}$*

The clause level matches the level of the last literal of the clause assigned to false.

The fundamental process to detect a generalized autarky modulo unit propagation consists in checking whether a clause is implied by unit propagation. The sequence of generated implications can be depicted by an acyclic oriented graph, called implication graph. Usually such a graph is used in all CDCL (conflict-driven clauses learning) approaches to learn clauses. It was first introduced in [17].

Definition 5 *An implication graph, noted $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ is a graph such that:*

1. *$\mathcal{V}$ is the set of vertices where each vertex, noted $s(l)$, is an assigned literal during unit propagation;*
2. *$\mathcal{A}$ is the set of edges. For a vertex $s(l_i)$ the set of predecessors is defined by the vertices $\{s(l_1), \ldots, s(l_{i-1}), s(l_{i+1}), \ldots, s(l_n)\}$ if there exists a clause $c = (\bar{l_1}, \ldots, \bar{l_{i-1}}, l_i, \bar{l_{i+1}}, \ldots, \bar{l_n})$ and a current partial assignment ρ such that $\{l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n\} \subseteq \rho$. Each arc $\langle s(l_j), s(l_i)\rangle$ is labeled by c.*

The roots of the implication graph representing the implication of a given clause by unit propagation are the vertices labelled by the negation of each literal of the clause. By construction, it is straightforward that this graph is acyclic.

Example 6 *Let Σ be the CNF formula built on the following 14 clauses:*

$c_1 : (\neg a \vee c)$	$c_6 : (\neg b \vee e)$	$c_{11} : (\neg i \vee l \vee \neg k)$
$c_2 : (\neg a \vee d)$	$c_7 : (\neg e \vee i)$	$c_{12} : (\neg i \vee \neg l)$
$c_3 : (\neg b \vee d)$	$c_8 : (\neg e \vee h)$	$c_{13} : (\neg k \vee \neg l)$
$c_4 : (\neg d \vee f)$	$c_9 : (\neg g \vee h \vee \neg f)$	$c_{14} : (c \vee e)$
$c_5 : (\neg d \vee g)$	$c_{10} : (\neg h \vee k)$	

The implication graph corresponding to the implication by unit propagation of the clause $(\bar{a} \vee \bar{b})$ is depicted in Fig. 3.

The goal is to compute the minimum levels of implication of the clauses that are used during the detection of the generalized local autarkies modulo unit propagation. So a value, noted $\eta(l)$, is associated to each vertex of the graph in the following way:

- Roots vertices have a null value;
- Let a vertex $s(l)$ and $\{s(l_{1,1}), \ldots, s(l_{1,m}), s(l_{2,1}), \ldots, s(l_{2,n}), \ldots, s(l_{k,1}), \ldots, s(l_{k,p})\}$ its set of predecessors such that $\langle s(l_{1,1}), s(l)\rangle, \ldots, \langle s(l_{1,m}), s(l)\rangle$ are labelled by c_1, $\langle s(l_{2,1}), s(l)\rangle, \ldots, \langle s(l_{2,n}), s(l)\rangle$ are labelled by $c_2, \ldots$, and $\langle s(l_{k,1}), s(l)\rangle, \ldots, \langle s(l_{k,p}), s(l)\rangle$ are labelled by c_k, then

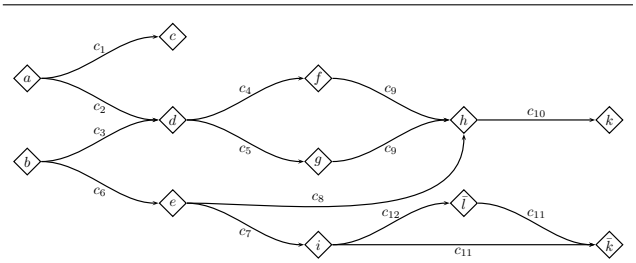

Figure 3. Implication graph representing an implication of a clause by unit propagation

$$\eta(s(l)) = \min[\max(\delta(c_1), \eta(s(l_{1,1})), \ldots, \eta(s(l_{1,m}))), \ldots, \max(\delta(c_k), \eta(s(l_{k,1})), \ldots, \eta(s(l_{k,p})))].$$

Example 7 *Let us consider the same CNF formula as in example 6 together with the following clause levels:*

$\delta(c_1) = 8$	$\delta(c_5) = 6$	$\delta(c_9) = 12$	$\delta(c_{13}) = 7$
$\delta(c_2) = 3$	$\delta(c_6) = 0$	$\delta(c_{10}) = 2$	$\delta(c_{14}) = 5$
$\delta(c_3) = 7$	$\delta(c_7) = 3$	$\delta(c_{11}) = 1$	
$\delta(c_4) = 9$	$\delta(c_8) = 3$	$\delta(c_{12}) = 4$	

For each vertex of the implication graph depicted in Fig. 3, the following values are obtained:

$\eta(s(a)) = 0$	$\eta(s(b)) = 0$	$\eta(s(c)) = 8$	$\eta(s(d)) = 3$
$\eta(s(e)) = 0$	$\eta(s(f)) = 9$	$\eta(s(g)) = 6$	$\eta(s(i)) = 3$
$\eta(s(h)) = 3$	$\eta(s(\bar{l})) = 4$	$\eta(s(k)) = 3$	$\eta(s(\bar{k})) = 4$

So, the implication level of the clause $(\bar{a}, \bar{b})$ is equal to $\max(\eta(s(k)), \eta(s(\bar{k}))) = 4$.

Thus, it is possible to compute an implication level for all the clauses used in the detection of generalized local autarkies modulo unit propagation. A level can be assigned to the detected autarky. It is defined as the greatest implication level computed for each implied clause. In consequence, if the level of the autarky is j, it is possible to backjump safely up to this decision level. Indeed, for a sub-formula Σ_i obtained from Σ and a partial assignment $\{l_0, \ldots, l_j, \ldots, l_i\}$, if all the clauses of Σ_i are implied modulo unit propagation at level $k \leq j$, then $\Sigma_j \vDash^* \Sigma_i$. All branches between j and the current level i are pruned.

Furthermore, even when no autarky is found, we have seen that the clauses that are proved to be implied during the autarky detection, are added to the formula. These clauses are added only until the current decision level is achieved. Since its minimal implication level has been computed, it is possible to keep the implied clauses until backtracking to their corresponding levels. This will reduce the size of search tree even more.

5. Conclusion

This paper presents a *formal framework* for the detection and use of autarky assignments, with the goal to im-

prove the efficiency of SAT solvers. Several contributions about the generalization of classical autarkies have been provided. The first one introduces the concept of autarky modulo unit propagation, which extends the usual definition by substituting the syntactical inclusion operation by a semantic-one based on unit propagation. It has also been proved that propagating unit literals are special cases of this generalization. A second extension introduces the concept of implication level, aiming at detecting larger partitions. Interestingly enough, when no generalized local autarky can be found, the approach is able to produce several sub-clauses from the formula. Each implied clause can be exploited in order to reduce the search space until backtracking to its corresponding implication level. Finally, it is shown how these different approaches could be integrated in DPLL-like SAT solvers.

As future works, we plan to investigate how to integrate and implement the approach in modern SAT solvers. This might prove very succesful for efficiently solving industrial-related SAT instances. It is well-known that many of these instances can be partitioned into several disconnected components after assigning a small set of variables. Another interesting path of research would be to design powerful clauses ordering heuristics for searching for local autarkies modulo unit propagation.

References

[1] Roberto J. Bayardo, Jr. and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, 1997.

[2] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1194–1201, 2003.

[3] Laure Brisoux, Éric Grégoire, and Lakhdar Saïs. Improving backtrack search for SAT by means of redundancy. In *International Syposium on Methodologies for Intelligent Systems*, pages 301–309, 1999.

[4] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[5] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3–SAT formulae. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 2001.

[6] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2002.

[7] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 431–437, 1998.

[8] Marijn J.H. Heule and Hans van Maaren. March dl: Adding adaptive heuristics and a new branching strategy. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:47–59, 2006.

[9] Jinbo Huang and Adnan Darwiche. A structure-based variable ordering heuristic for SAT. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1167–1172, 2003.

[10] Serge Jeannicot, Laurent Oxusoff, and Antoine Rauzy. évaluation sémantique en calcul propositionnel : une propriété de coupure pour rendre plus efficace la procédure de davis et putnam. *Revue d'Intelligence Artificielle*, 2(1):41–60, 1988. (in French).

[11] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[12] Henri Kautz, Eric Horvitz, Yongshao Ruan, Carla Gomes, and Bart Selman. Dynamic restart policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 674–682, 2002.

[13] Oliver Kullmann. Investigations on autark assignments. *Discrete Applied Mathematics*, 107(1-3):99–137, 2000.

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

[15] Wei Li and Peter van Beek. Guiding real-world SAT solving with dynamic hypergraph separator decomposition. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), 15-17 November 2004, Boca Raton, FL, USA*, pages 542–548, 2004.

[16] Mark H. Liffiton and Karem A. Sakallah. Searching for autarkies to trim unsatisfiable clause sets. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT'2008)*, volume 4996 of *LNCS*, pages 182–195. Springer Verlag, 2008.

[17] Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, 1996.

[18] Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.

[19] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[20] Laurent Oxusoff and Antoine Rauzy. *L'évaluation sémantique en calcul propositionnel*. Thèse de doctorat, Université de Provence, Marseille (France), January 1989.

[21] Allen Van Gelder. Autarky pruning in propositional model elimination reduces failure redundancy. *Journal of Automated Reasoning*, 23(2):137–193, 1999.

[22] Lintao Zhang. On subsumption removal and on-the-fly CNF simplification. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 482–489, 2005.