Pseudo-Propositional Logic

Ahmad-Saher Azizi-Sultan

Taibah University, Medinah Munawwarah, Saudi Arabia
sultansaher@hotmail.com

Abstract

Propositional logic is the main ingredient used to build up SAT solvers which have gradually become powerful tools to solve a variety of important and complicated problems such as planning, scheduling, and verifications. However further uses of these solvers are subject to the resulting complexity of transforming counting constraints into conjunctive normal form (CNF). This transformation leads, generally, to a substantial increase in the number of variables and clauses, due to the limitation of the expressive power of propositional logic. To overcome this drawback, this work extends the alphabet of propositional logic by including the natural numbers as a means of counting and adjusts the underlying language accordingly. The resulting representational formalism, called pseudo-propositional logic, can be viewed as a generalization of propositional logic where counting constraints are naturally formulated, and the generalized inference rules can be as easily applied and implemented as arithmetic.

1 Introduction

During the last few decades SAT solvers have gained considerable advances and become a tool suitable for attacking more and more practical problems arising in different areas such as formal verification [1, 2, 12], planning [9, 11], scheduling [8], etc. Most of these solvers, if not all, are a variety of Davis-Putnam-Logemann-Loveland (DPLL) algorithm [4, 5] which is based on blind branch and backtrack techniques that explore the search space exhaustively until a solution is found. As SAT is one of the canonical NP-complete problems [3], generally any exhaustive search algorithm results in impractical excessive time complexity.

In order to reduce the size of the search tree, modern SAT solvers such as Chaff, BerkMin, and MiniSAT have equipped the DPLL algorithm with pruning techniques known as backjumping, conflict-driven lemma learning, and restarts [6, 7, 10]. Although these techniques were able to reduce the search space, the major drawback of having blind control over the search process remains.

To mine solutions efficiently, there is a need for a tool that could scan over the search field and detect the spots that potentially contain solutions. Unfortunately, the limited expressive power of propositional logic does not allow for such a tool to be built-in. Furthermore, the input of SAT solvers is, usually, a formula in its CNF. However, many applications contain counting constraints and transforming these constraints into CNF generally leads to a substantial increase in the number of variables and clauses. This is again due to the lack of expressive tools in the underlying propositional language.

This work takes the liberty to extend the alphabet of propositional logic by including the natural numbers as a means of counting and adjusts the underlying language accordingly. The resulting representational formalism, which we can conveniently agree to call pseudo-propositional logic, may be viewed as a generalization of propositional logic, where counting constraints are naturally formulated and at the same time the Boolean nature of the propositional variables is kept preserved. This allows for encoding counting constraints as well as SAT instances much more compact than if it is encoded using CNF. Furthermore, the generalized inference rules are as easily applied and implemented as arithmetic. In such a case, equipping backtracking procedures with some combinatorial techniques allows for assigning truth values to a variety of propositional variables simultaneously. This leads to easily detecting the branches of the search tree that possibly contain solutions, or at least prune the useless ones, allowing for possible improvement in terms of calculation complexity.

2 Language

Definition 1. Let $\mathscr{P} = \{p, q, \cdots\}$ be a finite or countably infinite set of propositional symbols and $\mathbb{N}$ be the natural numbers. The alphabet $\mathscr{A}$ underlying the language of pseudo-propositional formulas is defined as $\mathscr{A} = \mathscr{P} \cup \mathbb{N} \cup \{\neg, +, (,)\}$, where $\{\neg, +, (,)\}$ resemble the negation, addition, opening and closing punctuation symbols, respectively.

Definition 2. The language or formulas of pseudo-propositional logic, symbolized by $\mathscr{F}$, is defined recursively as follows:

- If $p \in \mathscr{P}$ and $n \in \mathbb{N}$ then $np \in \mathscr{F}$, called prime formula.
- If $\alpha, \beta \in \mathscr{F}$, then $(\alpha + \beta)$, $\neg\alpha \in \mathscr{F}$.

A prime formula or its negation is called a *literal*. A formula which is a literal or an addition of two or more literals is said to be in *normal form*. A *subformula of a formula* α is a substring occurring in α, which is itself a formula.

Before proceeding further, let us agree upon the following two conventions to ease readability:

- Propositional variables or symbols will be denoted by $p, q, \ldots$, formulas by $\alpha, \beta, \varphi, \ldots$, set of formulas by $F, G, \ldots$, and set of queries, which will be defined in section 4, by $\boldsymbol{F}, \boldsymbol{G}, \ldots$, where these letters may also be indexed.
- As in arithmetical terms, parenthesis are omitted whenever it is possible.

3 Semantics

A proposition can have only one of the *truth values*, true or false. Conveniently to the context of this work, these values are represented by $(1,0)$ for true and $(0,1)$ for false. More formally, this is rephrased by the definition of interpretation.

Definition 3. An interpretation I is a subset of $\mathscr{P}$ represented by the mapping $\phi : \mathscr{P} \to \{(1,0), (0,1)\}$ which is defined as follows:

$$\phi(p) = \begin{cases} (1,0) & \text{if } p \in I, \\ (0,1) & \text{if } p \notin I. \end{cases}$$

Thus, the interpretation I is the subset of $\mathscr{P}$ containing only those propositional symbols that are mapped to $(1,0)$ under ϕ. That is

$$I = \{p \in \mathscr{P} \mid \phi(p) = (1,0)\}. \quad (1)$$

Recursively, the mapping ϕ can be extended to become from the set of formulas $\mathscr{F}$ to $\mathscr{M} = \mathbb{Z}^2$ which is the *meaning set* in the context of pseudo-propositional logic. In order to do so the following two functions are prerequisites:

- Negation $\neg^* : \mathscr{M} \to \mathscr{M}$ where $\neg^*(n,m) = (m,n)$.
- Addition[1] $+ : \mathscr{M} \times \mathscr{M} \to \mathscr{M}$, where $+((n,m),(k,l)) = (n+k, m+l)$.

Yet every interpretation I defines recursively its own mapping $I : \mathscr{F} \to \mathscr{M}$ as follows:

[1]This addition is easily distinguished from the addition of formulas in definition 1.

1) **Recursion base.** Recall that if φ is an atom then $\varphi = n\,p$ for some $n \in \mathbb{N}$ and $p \in \mathscr{P}$. Consequently the recursion base reads

$$I(\varphi) = I(n\,p) = n\,\phi(p).$$

2) **Recursion steps.**

$$I(\alpha) = \begin{cases} \neg^*(I(\beta)) & \text{if } \alpha \text{ is of the form } \neg\beta, \\ I(\beta_1) + I(\beta_2) & \text{if } \alpha \text{ is of the form } \beta_1 + \beta_2. \end{cases}$$

After assigning meanings to formulas, one can investigate how formulas are related to each other according to their meanings.

Definition 4. Two formulas α and β are *equivalent*, in symbols $\alpha \equiv \beta$, iff $I(\alpha) = I(\beta)$ for every interpretation I.

Example 1. *For any α, $\beta \in \mathscr{F}$, it is obvious that $\alpha + \beta \equiv \beta + \alpha$ and $\neg\neg\alpha \equiv \alpha$.*

Proposition 1. *For any α, $\beta \in \mathscr{F}$, the equivalence $\neg(\alpha + \beta) \equiv \neg\alpha + \neg\beta$ holds.*

Proof. Given an interpretation I suppose that $I(\alpha) = (n,l)$ and $I(\beta) = (m,k)$.

$$\begin{aligned} I(\neg(\alpha+\beta)) &= \neg^* I(\alpha+\beta) = \neg^*(I(\alpha) + I(\beta)) = \neg^*((n,l) + (m,k)) \\ &= \neg^*(n+m, l+k) = (l+k, n+m). \end{aligned}$$

On the other hand,

$$\begin{aligned} I(\neg\alpha + \neg\beta) &= I(\neg\alpha) + I(\neg\beta) = \neg^* I(\alpha) + \neg^* I(\beta)) = \neg^*(n,l) + \neg^*(m,k) \\ &= (l,n) + (k,m) = (l+k, n+m). \end{aligned}$$

Thus $I(\neg(\alpha+\beta)) = I(\neg\alpha + \neg\beta)$ for any given interpretation I. □

Proposition 2. *If $p \in \mathscr{P}$ and $n,m \in \mathbb{N}$ then $(n\,p + m\,p) \equiv (n+m)p$.*

Proof. Let I be an interpretation then,

$$I(np + mp) = I(np) + I(mp) = n\phi(p) + m\phi(p) = (n+m)\phi(p) = I((n+m)p).$$

□

Obviously, $\equiv$ is an equivalence relation. Moreover, it is a *congruence relation* on $\mathscr{F}$, i.e., for all $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathscr{F}$,

$$\alpha_1 \equiv \alpha_2, \beta_1 \equiv \beta_2 \Rightarrow \neg\alpha_1 \equiv \neg\alpha_2, \alpha_1 + \beta_1 \equiv \alpha_2 + \beta_2. \tag{2}$$

For this reason the *replacement theorem* holds. It enables one to substitute a subformula β, of a formula α, by an equivalent one without altering the meaning of α. If we let $\alpha\,[\beta_1/\beta_2]$ denote the formula that is obtained from α by substituting every occurrence of β_1 by β_2, the replacement theorem becomes as follows:

Theorem 1. *If the formulas β_1 and β_2 are equivalent, so are α and $\alpha\,[\beta_1/\beta_2]$.*

Proof by induction on α. Suppose α is a prime formula. Then, for both cases $\alpha = \beta_1$ and $\alpha \neq \beta_1$ we clearly have $\alpha \equiv \alpha\,[\beta_1/\beta_2]$. Now let $\alpha = \alpha_1 + \alpha_2$. If $\alpha = \beta_1$ then trivially $\alpha \equiv \alpha\,[\beta_1/\beta_2]$ holds. Otherwise $\alpha\,[\beta_1/\beta_2] = \alpha_1\,[\beta_1/\beta_2] + \alpha_2\,[\beta_1/\beta_2]$. By the induction hypothesis we have $\alpha_1 \equiv \alpha_1\,[\beta_1/\beta_2]$ and $\alpha_2 \equiv \alpha_2\,[\beta_1/\beta_2]$. According to the congruence property 2 one concludes that

$$\alpha = (\alpha_1 + \alpha_2) \equiv \alpha_1\,[\beta_1/\beta_2] + \alpha_2\,[\beta_1/\beta_2] = \alpha\,[\beta_1/\beta_2]\,.$$

The induction steps for $\neg$ follows analogously. □

Taking into account that one can eliminate all negation signs except those in front of prime formulas by Proposition 1, the replacement theorem transforms every formula into an equivalent one which is a normal form. This is actually interesting from an implementational point of view, as efficiency might be gained by restricting the inference rules to the mentioned normal form.

After assigning meaning to formulas and seeing how they are related, it is time to consider counting constraints which are represented by queries defined in the upcoming section.

4 Queries and Models

Definition 5. For a given formula $\varphi \in \mathscr{F}$ and an $n \in \mathbb{N}$, in pseudo-propositional logic we are interested in finding an answer to the query: is there an interpretation I such that $I(\varphi) = (m,l)$ where $m \geq n$. Every formula φ combined with a natural number n forms a query $\varphi^{(n)}$. The set of all possible queries is denoted by $\mathscr{Q}$. That is $\mathscr{Q} = \{\varphi^{(n)} : \varphi \in \mathscr{F}, n \in \mathbb{N}\}$.

Having defined queries, modelling becomes straightforward. Simply, it defines relations between interpretations and queries.

Definition 6. It is said that an interpretation I is a model for a query $\varphi^{(n)}$, in symbols $I \models \varphi^{(n)}$, iff $I(\varphi) = (\bar{n}, l)$ with $\bar{n} \geq n$. Considering a set of queries $\boldsymbol{Q}$, it is said that I is a model for $\boldsymbol{Q}$, and symbolised by $I \models \boldsymbol{Q}$, iff $I \models \varphi^{(n)}$ for every query $\varphi^{(n)} \in \boldsymbol{Q}$.

It is time now to start reasoning which is as easy as arithmetic. The coming proposition is an ideal example of reasoning in pseudo-propositional logic.

Proposition 3. *Let $\alpha^{(n)}, \varphi^{(m)} \in \mathscr{Q}$. If I is an interpretation such that $I \models \alpha^{(n)}$ and $I \models \varphi^{(m)}$, then $I \models (\alpha + \varphi)^{(n+m)}$.*

Proof. Since $I \models \alpha^{(n)}$ and $I \models \varphi^{(m)}$, this implies that $I(\alpha) = (\bar{n}, l)$, $\bar{n} \geq n$ and $I(\varphi) = (\bar{m}, k)$, $\bar{m} \geq m$. Thus

$$I(\alpha + \varphi) = I(\alpha) + I(\varphi) = (\bar{n}, l) + (\bar{m}, k) = (\bar{n} + \bar{m}, l + k).$$

Since $\bar{n} + \bar{m} \geq n + m$ we conclude that $I \models (\alpha + \varphi)^{(n+m)}$. □

One can easily conceive that satisfiability in pseudo-propositional logic concerns queries rather than formulas.

- It is said that $\varphi^{(n)}$ (resp. $\boldsymbol{Q}$) is *satisfiable* iff there exists an interpretation I such that $I \models \varphi^{(n)}$ (resp. $I \models \boldsymbol{Q}$).
- It is said that $\varphi^{(n)}$ (resp. $\boldsymbol{Q}$) is *unsatisfiable* iff for every interpretation I we have $I \not\models \varphi^{(n)}$ (resp. $I \not\models \boldsymbol{Q}$).

Consequently the equivalence relation is lifted to the level of queries as demonstrated below.

5 Consequence and Equivalence

Definition 7. $\boldsymbol{Q}$ is a logical consequence of $\boldsymbol{F}$, written $\boldsymbol{F} \models \boldsymbol{Q}$, if $I \models \boldsymbol{Q}$ for every interpretation I that is a model for $\boldsymbol{F}$. In short, $I \models \boldsymbol{F} \Rightarrow I \models \boldsymbol{Q}$ for all interpretations I.

Definition 8. If $\boldsymbol{F} \models \boldsymbol{Q}$ and $\boldsymbol{Q} \models \boldsymbol{F}$ then we say $\boldsymbol{F}$ and $\boldsymbol{Q}$ are semantically equivalent. We denote this by $\boldsymbol{F} \equiv \boldsymbol{Q}$.

In this work, $\boldsymbol{F} \models \alpha^{(n)}$ (resp. $\alpha^{(n)} \models \boldsymbol{F}$) will mean $\boldsymbol{F} \models \{\alpha^{(n)}\}$ (resp. $\{\alpha^{(n)}\} \models \boldsymbol{F}$). More generally, we write $\boldsymbol{F} \models \alpha_1^{(n_1)}, \alpha_2^{(n_2)}, \ldots, \alpha_k^{(n_k)}$ (resp. $\alpha_1^{(n_1)}, \alpha_2^{(n_2)}, \ldots, \alpha_k^{(n_k)} \models \boldsymbol{F}$) instead of $\boldsymbol{F} \models \{\alpha_1^{(n_1)}, \alpha_2^{(n_2)}, \ldots, \alpha_k^{(n_k)}\}$ (resp. $\{\alpha_1^{(n_1)}, \alpha_2^{(n_2)}, \ldots, \alpha_k^{(n_k)}\} \models \boldsymbol{F}$), and more briefly we write $\boldsymbol{F}, \alpha^{(n)} \models \varphi^{(m)}$ instead of $\boldsymbol{F} \cup \{\alpha^{(n)}\} \models \varphi^{(m)}$. Analogous notation will be used regarding semantic equivalence.

Note 1. *Proposition 3 can be rewritten as follows:*

$$\alpha^{(n)}, \varphi^{(m)} \models (\alpha + \varphi)^{(n+m)}.$$

Example 2. *Let* $\boldsymbol{Q} = \{\varphi^{(i)} : i = 1, 2, \ldots, n-1\}$*. Moreover suppose that* $I \models \varphi^{(n)}$*. This means that* $I(\varphi) = (\bar{n}, l)$ *where* $\bar{n} \geq n > i$*. Consequently,* $I \models \varphi^{(i)}$ *for all* $i < n$*. Thus* $\varphi^{(n)} \models \boldsymbol{Q}$*.*

Lemma 1. *Let* φ *be a formula and* $n > 1$ *then* $(\varphi + p + \neg p)^{(n)} \models \varphi^{(n-1)}$*.*

Proof. Suppose $I \models (\varphi + p + \neg p)^{(n)}$. This means that $I(\varphi + p + \neg p) = I(\varphi) + (1,1) = (\bar{n}, l)$ where $\bar{n} \geq n$ and $l \geq 1$. Since $(\mathbb{Z}^2, +)$ is an abelian group we conclude that $I(\varphi) = (\bar{n}-1, l-1)$. That is $I \models \varphi^{(n-1)}$. □

An obvious generalization and a direct consequence of lemma 1 is the resolution theorem which reads:

Theorem 2. *If* $\varphi \in \mathscr{F}$ *and* $n > \bar{m} \geq m$*, then the following consequence holds:*

$$(\varphi + \bar{m}p + \neg(mp))^{(n)} \models (\varphi + (\bar{m} - m)p)^{(n-m)}.$$

To keep things from being complicated before going any further, the following theorem must be proven.

Theorem 3. *If the formulas* β_1 *and* β_2 *are equivalent, so are the queries* $\alpha^{(n)}$ *and* $(\alpha\,[\beta_1/\beta_2])^{(n)}$ *for every* $n \in \mathbb{N}$*.*

Proof. Suppose that $I \models \alpha^{(n)}$. This means that $I(\alpha) = (\bar{n}, l)$ where $\bar{n} \geq n$. Since $\beta_1 \equiv \beta_2$, from Theorem 1 it follows that $\alpha \equiv \alpha\,[\beta_1/\beta_2]$. Thus $I(\alpha) = I(\alpha\,[\beta_1/\beta_2]) = (\bar{n}, l)$ where $\bar{n} \geq n$. Thus $I \models (\alpha\,[\beta_1/\beta_2])^{(n)}$. We conclude that $\alpha^{(n)} \models (\alpha\,[\beta_1/\beta_2])^{(n)}$. Taking into account that $(\alpha\,[\beta_1/\beta_2])\,[\beta_2/\beta_1] = \alpha$, the consequence $(\alpha\,[\beta_1/\beta_2])^{(n)} \models \alpha^{(n)}$ follows analogously. □

If we call a query with a normal form formula a *normal form query*, then theorems 1 and 3 transform any query into an equivalent one which is a normal form. Thus to solve the satisfiability problem in pseudo-proposition logic it suffices to consider only the sets of queries that are normal form.

Finally, to start applying pseudo-propositional logic to solve real-world problems, such as SAT for example, one more theorem is needed.

Theorem 4. *If* $\boldsymbol{F} \models \boldsymbol{Q}$ *and for every interpretation* I *which models* $\boldsymbol{Q}$ *we have* $I \not\models \boldsymbol{F}$*, then* $\boldsymbol{F}$ *is unsatisfiable.*

Proof. Suppose that $\boldsymbol{F}$ is satisfiable. This means that there exists an interpretation I such that $I \models \boldsymbol{F}$. Consequently, $I \models \boldsymbol{Q}$ since $\boldsymbol{F} \models \boldsymbol{Q}$. This contradicts the theorem's hypothesis. $\square$

Informally speaking, before solving a SAT problem for a given set of queries $\boldsymbol{F}$, Theorem 4 tempts one to use proper inference rules to find a simpler set of queries $\boldsymbol{Q}$ such that $\boldsymbol{F} \models \boldsymbol{Q}$. Now it is enough to look for a solution for $\boldsymbol{F}$ among only those solutions that solve $\boldsymbol{Q}$.

Furthermore any algorithm that solves SAT problem in pseudo-propositional logic can be used to solve the SAT problem of propositional logic. Actually pseudo-propositional logic can be viewed as a generalization of propositional logic. This is justified by the fact that every formula or sentence S in propositional logic can be represented equivalently by a set of queries $\boldsymbol{Q}$ in pseudo-propositional logic. To see this let $\mathscr{P} = \{p_1, p_2, p_3, \cdots\}$ be our set of propositional symbols and let the literal l_i be either p_i or its negation. Moreover, suppose that converting the sentence S into its CNF results in the set of clauses $\{C_i : i = 1, 2, \ldots, n\}$ where each clause C_i is the disjunctions of a set of literals $\{l_j : j \in J_i \subset \mathbb{N}\}$. If we denote to S in its CNF by S_{CNF} we conclude that

$$S_{CNF} = \bigwedge_{i=1}^{n} C_i = \bigwedge_{i=1}^{n} \left(\bigvee_{j \in J_i} l_j \right).$$

Clearly each clause C_i which has the form

$$C_i = \bigvee_{j \in J_i} l_j$$

can be equivalently represented in pseudo-propositional logic by the query

$$Q_i = \left(\sum_{j \in J_i} l_j \right)^{(1)}.$$

If we let $\boldsymbol{Q} = \{Q_i : i = 1, 2, \ldots, n\}$ then the sentence S, which is equivalent to S_{CNF}, is satisfiable iff $\boldsymbol{Q}$ is satisfiable. Moreover, an interpretation I is a model for $\boldsymbol{Q}$ iff I is a model for S. A detailed example is presented in the sequel.

6 Application on SAT

This section is not meant to present an algorithm that competes with the current SAT solvers. It just gives an idea of how one can make use of pseudo-propositional logic to solve problems such as SAT. This is actually done in three steps. Intuitively, the first step involves transforming the given SAT instance into the corresponding set of queries $\boldsymbol{F}$ in pseudo-propositional logic. The second step reprocesses the set $\boldsymbol{F}$ using inference rules to generate a proper compact set of queries $\boldsymbol{Q}$ such that $\boldsymbol{F} \models \boldsymbol{Q}$. Finally, apply a backtracking procedure to find a solution for $\boldsymbol{Q}$ and check if it satisfies the original SAT instance. The final step is repeated until a solution is found or the problem is unsatisfiable otherwise.

Example 3. *Consider the following CNF instance:*

$$\begin{gathered}
x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee \neg x_2,\ \neg x_1 \vee \neg x_3,\ \neg x_2 \vee \neg x_3, \\
x_1 \vee x_2 \vee x_4,\ \neg x_1 \vee \neg x_2,\ \neg x_1 \vee \neg x_4,\ \neg x_2 \vee \neg x_4, \\
x_1 \vee x_3 \vee x_4,\ \neg x_1 \vee \neg x_3,\ \neg x_1 \vee \neg x_4,\ \neg x_3 \vee \neg x_4, \\
x_2 \vee x_3 \vee x_4,\ \neg x_2 \vee \neg x_3,\ \neg x_2 \vee \neg x_4,\ \neg x_3 \vee \neg x_4.
\end{gathered}$$

This can be equivalently represented by a set of queries in pseudo-propositional logic as follows:

$$\begin{aligned}\boldsymbol{F} = \{ \quad & (x_1+x_2+x_3)^{(1)}, (\neg x_1+\neg x_2)^{(1)}, (\neg x_1+\neg x_3)^{(1)}, (\neg x_2+\neg x_3)^{(1)},\\ & (x_1+x_2+x_4)^{(1)}, (\neg x_1+\neg x_2)^{(1)}, (\neg x_1+\neg x_4)^{(1)}, (\neg x_2+\neg x_4)^{(1)},\\ & (x_1+x_3+x_4)^{(1)}, (\neg x_1+\neg x_3)^{(1)}, (\neg x_1+\neg x_4)^{(1)}, (\neg x_3+\neg x_4)^{(1)},\\ & (x_2+x_3+x_4)^{(1)}, (\neg x_2+\neg x_3)^{(1)}, (\neg x_2+\neg x_4)^{(1)}, (\neg x_3+\neg x_4)^{(1)}\}.\end{aligned}$$

Taking into account Proposition 2 while adding each consecutive homogeneous[2] *couple of queries in* $\boldsymbol{F}$ *results in*

$$\begin{aligned}\boldsymbol{F}_1 = \{ \quad & (2x_1+2x_2+x_3+x_4)^{(2)}, (\neg 2x_1+\neg x_2+\neg x_3)^{(2)}, (\neg x_1+\neg 2x_2+\neg x_3)^{(2)},\\ & (\neg x_1+\neg x_2+\neg 2x_4)^{(2)}, (x_1+x_2+2x_3+2x_4)^{(2)}, (\neg 2x_1+\neg x_3+\neg x_4)^{(2)},\\ & (\neg x_2+\neg 2x_3+\neg x_4)^{(2)}, (\neg x_2+\neg x_3+\neg 2x_4)^{(2)}\}.\end{aligned}$$

One can easily conceive that $\boldsymbol{F} \equiv \boldsymbol{F}_1$ *which shows how encoding in pseudo-propositional logic is much more compact than it is in CNF. Although it is not always the case that* $\boldsymbol{F} \equiv \boldsymbol{F}_1$ *but we, at least, know from Proposition 3 that* $\boldsymbol{F} \models \boldsymbol{F}_1$*. If we add again and again each consecutive homogeneous couple of queries in* $\boldsymbol{F}_1$*, we finally get*

$$\boldsymbol{F}_2 = \{(3x_1+3x_2+3x_3+3x_4)^{(4)}, (\neg 6x_1+\neg 6x_2+\neg 6x_3+\neg 6x_4)^{(12)}\},$$

as a logical consequence of $\boldsymbol{F}_1$*. By backtracking and propagation,* $\boldsymbol{F}_2$ *has only the following six interpretations:*

$$\begin{aligned}I_1 = \{x_1,x_2\}, I_2 = \{x_1,x_3\}, I_3 = \{x_1,x_4\},\\ I_4 = \{x_2,x_3\}, I_5 = \{x_2,x_4\}, I_6 = \{x_3,x_4\}.\end{aligned}$$

Since non of these interpretations model $\boldsymbol{F}$*, according to Theorem 4* $\boldsymbol{F}$ *and consequently the original SAT instance are unsatisfiable.*

7 Conclusion and Further Work

This work has introduced pseudo-propositional logic, a generalization of propositional logic with considerable extension of its expressive power. This enables the resulting representational formalism to encode counting constraints as well as SAT instances naturally, and much more compact than the encoding using CNF. The inference rules of the resulting pseudo-propositional logic, besides their Boolean nature, have arithmetical flavour allowing for easy implementation. Moreover, as it was seen in Example 3, applying backtracking on the final entailed set of queries may yield simultaneous multi-variables guesses, eliminating considerable parts of the search tree that have not been yet explored and hence allowing for potential improvement in terms of time complexity. Another promising improvement is subject to a further investigation on how to construct a compact proper final entailed set of queries which maximizes the eliminated part of the search tree and at the same time captures all possible solutions of the original problem.

8 Acknowledgments

I would like to thank Prof. Steffen Hölldobler, Director of the International Center for Computational Logic, Dresden, Germany. I have learned from him how to rationalise logically rather than just thinking mathematically. Without his influence this work would not have been established.

[2]containing literals of the same polarity

References

[1] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, DAC '99, pages 317–320, New York, NY, USA, 1999. ACM.

[2] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, UK, 1999. Springer-Verlag.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971.

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

[5] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[6] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518, 2003.

[7] Eugene Goldberg and Yakov Novikov. Berkmin: A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12):1549 – 1561, 2007. SAT 2001, the Fourth International Symposium on the Theory and Applications of Satisfiability Testing.

[8] Carla P. Gomes, Bart Selman, Ken McAloon, and Carol Tretkoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburgh, Pennsylvania, USA, 1998*, pages 208–213, 1998.

[9] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, pages 1194–1201. AAAI Press, 1996.

[10] 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 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535, 2001.

[11] Stuart J. Russell and Peter Norvig. *Artificial intelligence - a modern approach, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, 2003.

[12] Miroslav N Velev and Randal E Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. *Journal of Symbolic Computation*, 35(2):73 – 106, 2003.