Notes on Pseudo-Boolean Implication Proofs Version of December 9, 2022

Randal E. Bryant

Computer Science Department
Carnegie Mellon University, Pittsburgh, PA, United States
Randy.Bryant@cs.cmu.edu

This notes describes some ideas on converting unsatisfiability proofs for pseudo-Boolean (PB) constraints into clausal proofs based on the DRAT proof system.

1 Notation

This section gives brief definitions of pseudo-Boolean constraints and their properties. A more extensive introduction is provided by Gocht in his PhD thesis [Gocht-Phd-2022].

Consider a set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$. For $x_i \in X$, *literal* ℓ_i can denote either x_i or its negation, written $\overline{x}_i$. We write $\overline{\ell}_i$ to denote $\overline{x}_i$ when $\ell_i = x_i$ and x_i when $\ell_i = \overline{x}_i$.

A *pseudo-Boolean constraint* is a linear expression, viewing Boolean variables as ranging over integer values 0 and 1. That is, for a set of indices $I_C \subseteq \{1, 2, \ldots, n\}$, a constraint C has the form

$$\sum_{i \in I_C} a_i \ell_i \quad \# \quad b \tag{1}$$

where: 1) the relational operator $\#$ is $<$, $\leq$, $=$, $\geq$, or $>$, and 2) the coefficients a_i, as well as the constant b, are integers, with the coefficients being nonzero. Constraints with relational operator $=$ are referred to as *equational constraints*, while others are referred to as *ordering constraints*.

Constraint C denotes a Boolean function, written $[\![C]\!]$, mapping assignments to the set of variables X to 1 (true) or 0 (false). Constraints C_1 and C_2 are said to *equivalent* when $[\![C_1]\!] = [\![C_2]\!]$. Constraint C is said to be *infeasible* when $[\![C]\!] = \bot$, i.e., it always evaluates 0. C is said to be *trivial* when $[\![C]\!] = \top$, i.e., it always evaluates to 1.

As described in [Gocht-Phd-2022], the following are some properties of pseudo-Boolean constraints:

- Constraints with relational operators $<$, $\leq$, and $>$ can be converted to equivalent constraints with relational operator $\geq$.

- The logical negation of an ordering constraint C, written $\neg C$, can also be expressed as a constraint. That is, assume that C has been converted to a form where it has relational operator $\geq$. Then replacing $\geq$ by $<$ yields the negation of C.

- A constraint C with relational operator $=$ can be converted to the pair of constraints $C_{\leq}$ and $C_{\geq}$, formed by replacing $\#$ in (1) by $\leq$ and $\geq$, respectively. These then satisfy $[\![C]\!] = [\![C_{\leq}]\!] \wedge [\![C_{\geq}]\!]$.

We will generally assume that constraints have relational operator $\geq$, since other forms can be translated to it. In some contexts, however, we will maintain equational constraints intact, to avoid replacing it by two ordering constraints.

We consider two *normalized forms* for ordering constraints: A *coefficient-normalized* constraint has only positive coefficients. A *variable-normalized* constraint has only positive literals. Converting between the two forms is straightforward using the identity $\overline{x}_i = 1 - x_i$. In reasoning about PB constraints, the two forms can be used interchangeably. Typically, the coefficient-normalized form is more convenient when viewing a PB constraint as a logical expression, while the variable-normalized form is more convenient when viewing a constraint as an arithmetic expression. In this document, we focus on the logical aspects, giving the general form of constraint C as

$$\sum_{i \in I_C} a_i \ell_i \quad \geq \quad b \tag{2}$$

with each $a_i > 0$.

Ordering constraint C in coefficient-normalized form is trivial if and only if $b \leq 0$. Similarly, C is infeasible if and only if $b > \sum_{i \in I_C} a_i$. By contrast, testing feasiblilty or triviality of an equational constraint is not straightforward, in that an instance of the subset sum problem [Garey] can be directly encoded as an equational constraint.

An *assignment* is a mapping $\rho : X' \rightarrow \{0, 1\}$, for some $X' \subseteq X$. The assignment is *total* when $X' = X$ and *partial* when $X' \subset X$. Assignment ρ can also be viewed as a set of literals, where $x_i \in \rho$ when $\rho(x_i) = 1$ and $\overline{x}_i \in \rho$ when $\rho(x_i) = 0$. Finally, assignment ρ can be viewed as the pseudo-Boolean constraint $\sum_{l_i \in \rho} l_i \;\geq\; |\rho|$. We use these three views interchangeably. Note also for assignments ρ_1 and ρ_2 over disjoint sets of variables X_1 and X_2, their union is logically equivalent to their conjunction: $[\![\rho_1 \cup \rho_2]\!] = [\![\rho_1]\!] \wedge [\![\rho_2]\!]$.

We let $\rho(C)$ denote the constraint resulting when C is simplified according assignment ρ. That is, assume ρ is defined over a set of variables X' and partition the indices $i \in I_C$ into three sets: I^+, consisting of those indices i such that $\ell_i \in \rho$, I^-, consisting of those indices i such that $\overline{\ell}_i \in \rho$, and I^X consisting of those indices i such that $x_i \notin X'$. With this, $\rho(C)$ can be written

$$\sum_{i \in I^X} a_i \ell_i \quad \geq \quad b - \sum_{i \in I^+} a_i \tag{3}$$

A pseudo-Boolean *formula* F is a set of pseudo-Boolean constraints. We say that F is *satisfiable* when there is some assignment ρ that satisfies all of the constraints in F, and *unsatisfiable* otherwise.

2 Pseudo-Boolean Implication Proofs

A Pseudo-Boolean Implication Proof (PBIP) provides a systematic way to prove that a PB formula F is unsatisfiable. It consists of a sequence of ordering constraints

$$C_1, C_2, \ldots, C_m, C_{m+1}, \ldots, C_t$$

such that the first m constraints are those of formula F, while the constraints i with $i > m$ are implied by either one or two of the preceding constraints. That is, for each step i with $m < i \leq t$, there is a set $H_i \subseteq \{C_1, C_2, \ldots, C_{i-1}\}$ such that $|H_i| \leq 2$ and

$$\bigwedge_{C \in H_i} [\![C]\!] \quad \Rightarrow \quad [\![C_i]\!] \tag{4}$$

The proof completes with an infeasible constraint C_t. By the transitivity of implication, we have therefore proved that F is not satisfiable. We refer to the set H_i as the *hints* for step i.

Unless $P = NP$, we cannot guarantee that a proof checker can validate even a single step of a PBIP proof in polynomial time. In particular, consider an equational constraint C encoding an instance of the subset sum problem, and let $C_{\leq}$ and $C_{\geq}$ denote its conversion into a pair of ordering constraints such that $\llbracket C \rrbracket = \llbracket C_{\leq} \rrbracket \wedge \llbracket C_{\geq} \rrbracket$. Consider a PBIP proof step to add the constraint $\neg C_{\leq}$ having the set $\{C_{\geq}\}$ as hints. Proving that $\llbracket C_{\geq} \rrbracket \Rightarrow \llbracket \neg C_{\leq} \rrbracket$, requires proving that $\llbracket C_{\leq} \rrbracket \wedge \llbracket C_{\geq} \rrbracket = \bot$, i.e., that C is unsatisfiable.

On the other hand, checking the correctness of a PBIP proof can be performed in *pseudo-polynomial* time, meaning that the complexity will be bounded by a polynomially sized formula over the numeric values of the integer parameters. This can be done using binary decision diagrams [BBH-2022]. In particular, an ordering constraint over n variables in coefficient-normalized form with constant b will have a BDD representation with at most $b \cdot n$ nodes. For proof step where the added clause and the hints all have constants less than or equal to b, the number of BDD operations to validate the step will be $O(b^2 \cdot n)$ when there is a single hint and $O(b^3 \cdot n)$ when there are two hints. This complexity is polynomial in b, but it would be exponential in the size of a binary representation of b.

3 (Reverse) Unit Propagation

Consider constraint C in coefficient-normalized form. Literal ℓ is *unit propagated* by C when the assignment $\rho = \{\overline{\ell}\}$ causes the constraint $\rho(C)$ to become infeasible. Observe that a single constraint can unit propagate multiple literals. For example, $4x_1 + 3\overline{x}_2 + x_3 \geq 6$ unit propagates both x_1 and $\overline{x}_2$. For constraint C, we let $\beta[C]$ denote the set of literals it unit propagates. This set can also be viewed as a partial assignment, and therefore also a constraint, having the property that $\llbracket C \rrbracket \Rightarrow \llbracket \beta[C] \rrbracket$.

For an ordering constraint C in coefficent-normalized form (2), detecting which literals unit propagate is straightforward. Let $A = \sum_{i \in I_C} a_i$. Then literal ℓ_i unit propagates if and only if $A - a_i < b$, i.e., $a_i > A - b$. For example, the constraint $4x_1 + 3\overline{x}_2 + x_3 \geq 6$ has $A = 7$ and $b = 6$, yielding $A - b = 1$. This justifies the unit propagations of both x_1 and $\overline{x}_2$. A constraint of the form $\ell \geq 1$ is the constraint representation of a unit clause—it will unit propagate ℓ.

To perform unit propagation on constraint C in the context of a partial assignment α, we can use (3) to first generate the constraint representation of $\rho(C)$.

The *reverse unit propagation* (RUP) proof rule [Gocht-Phd-2022] uses unit propagation to prove that constraint C can be added to formula F while preserving its set of satisfying assignments. It operates by finding a sequence of constraints $C_0, C_1, \ldots, C_k$, with $C_0 = \neg C$ and $C_i \in F$ for $1 \leq i \leq k$, and proving that this combination is unsatisfiable.

A successful RUP proof generates a sequence of assignments $\rho_0, \rho_1, \ldots, \rho_{k-1}$ with $\rho_0 = \beta[C_0]$ and $\rho_i = \rho_{i-1} \cup \beta[\rho_{i-1}(C_i)]$ for $1 < i < k$. That is, it extends the set of unit literals with those obtained by first applying the unit literals to C_i and then performing unit propagation. The final clause C_k must then satisfy $\rho_{k-1}(C_k) = \bot$.

Given a RUP proof step, we can generate a sequence of PBIP proof steps that concludes with the addition of clause C. In particular, the final assignment in the RUP proof yielded the result $\llbracket \rho_{k-1}(C_k) \rrbracket = \llbracket \rho_{k-1} \rrbracket \wedge \llbracket C_k \rrbracket \Rightarrow \bot$, and therefore $\llbracket C_k \rrbracket \Rightarrow \llbracket \neg \rho_{k-1} \rrbracket$. Continuing for $1 \leq i < k$, we can see $\llbracket \rho_i \rrbracket = \llbracket \rho_{i-1} \rrbracket \wedge \llbracket C_i \rrbracket$, but the previous step added $\neg \rho_i$, and therefore we can add the clause $\neg \rho_{i-1}$, with the preceding clause plus clause C_i as hints. Once we reach $i = 1$, we have added the clause $\neg \rho_0$, and we can therefore add clause C with the preceding clause as its hint, since $\llbracket C_0 \rrbracket \Rightarrow \llbracket \rho_0 \rrbracket$, and therefore $\llbracket \neg \rho_0 \rrbracket \Rightarrow \llbracket \neg C_0 \rrbracket = \llbracket C \rrbracket$.

Summarizing, the PBIP proof steps will be as follows:

Added Clause	Hints
$\neg\rho_{k-1}$	C_k
$\neg\rho_{k-2}$	C_{k-1}, $\neg\rho_{k-1}$
$\cdots$	$\cdots$
$\neg\rho_i$	C_{i+1}, $\neg\rho_{i+1}$
$\cdots$	$\cdots$
$\neg\rho_0$	C_1, $\neg\rho_1$
C	$\neg\rho_0$

Note in these that the constraint representation of an assignment or its inverse has a very simple form. In particular, if $\rho = \{\ell_1, \ell_2, \ldots, \ell_k\}$, then its contstraint representation is $\ell_1 + \ell_2 + \cdots + \ell_k \geq k$. On the other hand, constraint $\neg\rho$ can be written as $\overline{\ell}_1 + \overline{\ell}_2 + \cdots + \overline{\ell}_k \geq 1$.

4 RUP Example

As an example, consider the following three clauses:

ID	Constraint
C_1	$x_1 + 2x_2 + \overline{x}_3 \geq 2$
C_2	$\overline{x}_1 + \overline{x}_2 + 2x_3 \geq 2$
C_3	$x_1 + 2\overline{x}_2 + 3\overline{x}_3 \geq 3$

Our goal is to add the clause $C = 2x_1 + x_2 + x_3 \geq 2$ via RUP using these clauses. RUP proceeds as follows:

1. We can see that $\neg C = 2\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 3$, and this unit propagates assignment $\rho_0 = \overline{x}_1 \geq 1$.
2. With this, constraint C_1 simplifies to $2x_2 + \overline{x}_3 \geq 2$, and this unit propagates x_2, giving $\rho_1 = \overline{x}_1 + x_2 \geq 2$.
3. Constraint C_2 simplifies to $2x_3 \geq 1$, which unit propagates x_3, giving $\rho_2 = \overline{x}_1 + x_2 + x_3 \geq 3$.
4. Constraint C_3 simplifies to $0 \geq 3$, which is infeasible.

This single application of RUP results in the following PBIP proof steps:

ID	Constraint	Hints
C_4	$x_1 + \overline{x}_2 + \overline{x}_3 \geq 1$	C_3
C_5	$x_1 + \overline{x}_2 \geq 1$	C_2, C_4
C_6	$x_1 \geq 1$	C_1, C_5
C	$2x_1 + x_2 + x_3 \geq 2$	C_6