Notes on
Pseudo-Boolean Implication Proofs
Version of April 13, 2023

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. For $x \in X$, *literal* ℓ can denote either x or its negation, written $\overline{x}$. We write $\overline{\ell}$ to denote $\overline{x}$ when $\ell = x$ and x_i when $\ell = \overline{x}$.

A *pseudo-Boolean constraint* is a linear expression, viewing Boolean variables as ranging over integer values 0 and 1. That is, a constraint C has the form

$$\sum_{1 \leq i \leq n} a_i \ell_i \quad \# \quad b \tag{1}$$

where: 1) the relational operator $\#$ is $<$, $\leq$, $=$, $\geq$, or $>$, and 2) the coefficients a_i, as well as the constant b, are integers. 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 $\overline{C}$, 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 $[\![C]\!] = [\![C_{\leq}]\!] \land [\![C_{\geq}]\!]$.

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 nonnegative coefficients. By convention, we require with this form that literal $\ell_i = x_i$ for any i such that $a_i = 0$. 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

$$\sum_{1 \leq i \leq n} a_i \ell_i \quad \geq \quad b \tag{2}$$

with each $a_i \geq 0$, and with $\ell_i = x_i$ whenever $a_i = 0$.

Ordering constraint C in coefficient-normalized form is trivial if and only if $b \leq 0$. Similarly, C is infeasible if and only if $b > \sum_{1 \leq i \leq n} a_i$. By contrast, testing feasibility 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 $\rho(x_i) = 0$. A total assignment

Some nomenclature regarding constraints of the form of (2) will prove useful. The *constraint literals* are those literals ℓ_i such that $a_i \neq 0$. A *cardinality constraint* has $a_i \in \{0, 1\}$ for $1 \leq i \leq n$. A cardinality constraint with $b = 1$ is referred to as a *clause*: at least one of the constraint literals must be assigned 1 to satisfy the constraint. A cardinality constraint with $\sum_{1 \leq i \leq n} a_i = b$ is referred to as a *conjunction*: all of the constraint literals must be assigned 1 to satisfy the constraint. Observe that any assignment ρ can be viewed as a conjunction, having coefficient $a_i = 1$ for each $\ell_i \in \rho$. A conjunction with for which $a_i = 1$ for just a single value of i is referred to as a *unit* constraint: it is satisfied if and only if literal ℓ_i is assigned 1.

We let $C|_\rho$ 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 for $1 \leq i \leq n$ 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, $C|_\rho$ can be written as $\sum_{1 \leq i \leq n} a'_i \;\; \geq \;\; b'$ with a'_i equal to a_i for $i \in I^X$ and equal to 0 otherwise, and with $b' = b - \sum_{i \in I^+} a_i$.

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 is given by a sequence of constraints, referred to as the *proof sequence*:

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

such that the first m constraints are those of formula F, while each *added* constraint C_i for $i > m$ follows by implication from the preceding constraints. That is,

$$\bigwedge_{1 \leq j \leq i-1} [\![C_j]\!] \quad \Rightarrow \quad [\![C_i]\!] \tag{3}$$

The proof completes with the addition of an infeasible constraint for C_t. By the transitivity of implication, we have therefore proved that F is not satisfiable.

Constraints C_i with $i > m$, can be added in two different ways, corresponding to two different reasoning modes.

1. In *implication mode*, constraint C_i follows by implication from at most two prior constraints in the proof sequence. That is, for some $H_i \subseteq \{C_1, C_2, \ldots, C_{i-1}\}$ with $|H_i| \leq 2$ such that:

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

   Set H_i is referred to as the *hint* for proof step i.

2. In *counterfactual* mode, the new constraint is justified via a proof by contradiction. That is, C_i is introduced as a *target constraint* and is followed by a sequence of *counterfactual constraints* $D^i_1, D^i_2, \ldots, D^i_k$ that would hold if the target constraint were false. (These constraints are not part of the proof sequence.) The final constraint D^i_k is infeasible, thus showing by contradiction that the target constraint must hold. Each step in the counterfactual sequence must follow by implication from one or two constraints as a hint, where the hint H^i_j for D^i_j can contain the following:

   (a) A constraint $C_{i'}$ from the proof sequence.

   (b) The negation of the target constraint $\overline{C}_i$

   (c) An earlier counterfactual constraint within the same sequence $D^i_{j'}$ with $j' < j$.

   At least one of the constraints in the hint must be from the latter two categories. While in counterfactual mode, the proof can temporarily revert to implication mode, adding a constraint $C_{i'}$ to the proof sequence that follows by implication from at most two other constraints in the proof sequence. Once the final counterfactual constraint D^i_k is given, the proof reverts to implication mode, and constraint C_i can be used as the antecedent in subsequent proof steps.

Unless $P = NP$, we cannot guarantee that a proof checker can validate even a single implication 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 $[\![C]\!] = [\![C_{\leq}]\!] \wedge [\![C_{\geq}]\!]$. Consider a PBIP proof step to add the constraint $\overline{C}_{\leq}$ having the $C_{\geq}$ as the only hint. Proving that $[\![C_{\geq}]\!] \Rightarrow [\![\overline{C}_{\leq}]\!]$, requires proving that $[\![C_{\leq}]\!] \wedge [\![C_{\geq}]\!] = \bot$, 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 constraints 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 $C|_\rho$ to become infeasible. As the name implies, a unit-propagated literal ℓ then becomes a unit constraint. 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 an ordering constraint C in coefficent-normalized form (2), detecting which literals unit propagate is straightforward. Let $A = \sum_{1 \leq i \leq n} a_i$. 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 $\overline{x}_2$.

For constraint C, we let $\mathit{Unit}(C)$ denote the set of literals it unit propagates. Often, by simplifying a constraint C according to a partial assignment ρ, the simplified constraint $C|_\rho$ will unit propagate new literals, given by $\mathit{Unit}(C|_\rho)$. These literals can then be added to the partial assignment. Formally, define the operation *Uprop* as $\mathit{Uprop}(\rho, C) = \rho \cup \mathit{Unit}(C|_\rho)$. *Unit propagation* is then the process of repeatedly applying this operation to a set of clauses to expand the set of literals in a partial assignment.

The *reverse unit propagation* (RUP) proof rule [Gocht-Phd-2022] uses unit propagation to prove that constraint can be added to formula while preserving its set of satisfying assignments. It closely parallels the counterfactual reasoning mode of PBIP, using unit propagation as the basic step for generating the counterfactual constraints. When adding clause C_i, each constraint D^i_j in the counterfactual sequence represents the conjunction of a set of literals ρ_j. Let ρ_0 denote the conjunction of all literals that have been derived as unit constraints previously. For each j with $1 \leq j < k$, counterfactual assertion ρ_j is derived either from some previous proof clause: $\rho_j = \mathit{Uprop}(\rho_{j-1}, C_j)$ for $j < i$, or from the negated target: $\rho_j = \mathit{Uprop}(\rho_{j-1}, \overline{C}_i)$. The final contradiction occurs when $C_k|_{\rho_{k-1}}$ is infeasible.

4 RUP Example

As an example, consider the following three constraints:

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

Our goal is to add the constraint $C = 2x_1 + x_2 + x_3 \geq 2$ via RUP using these constraints, without any prior unit constraints. RUP proceeds as follows:

1. We can see that $\overline{C} = 2\overline{x}_1 + \overline{x}_2 + \overline{x}_3 \geq 3$, and this unit propagates assignment $\rho_1 = \overline{x}_1 \geq 1$.
2. With this, constraint C_1 simplifies to $2x_2 + \overline{x}_3 \geq 2$, and this unit propagates x_2, giving $\rho_2 = \overline{x}_1 + x_2 \geq 2$.
3. Constraint C_2 simplifies to $2x_3 \geq 1$, which unit propagates x_3, giving $\rho_3 = \overline{x}_1 + x_2 + x_3 \geq 3$.
4. Constraint C_3 simplifies to $0 \geq 3$, which is infeasible.