Logic Programming with
Pseudo-Boolean Constraints

Alexander Bockmayr

MPI-I-91-227        April/December 1991
Author’s Address

Alexander Bockmayr, Max-Planck-Institut für Informatik, Im Stadtwald, D-6600 Saarbrücken, bockmayr@mpi-sb.mpg.de

Publication Notes

The present report has been submitted for publication elsewhere and will be copyrighted if accepted.

Acknowledgements

I would like to thank Peter Barth, David Basin and Harald Ganzinger for their comments on a preliminary version of this paper.
Abstract

Boolean constraints play an important role in various constraint logic programming languages. In this paper we consider pseudo-Boolean constraints, that is equations and inequalities between pseudo-Boolean functions. A pseudo-Boolean function is an integer-valued function of Boolean variables and thus a generalization of a Boolean function. Pseudo-Boolean functions occur in many application areas, in particular in problems from operations research. An interesting connection to logic is that inference problems in propositional logic can be translated into linear pseudo-Boolean optimization problems. More generally, pseudo-Boolean constraints can be seen as a particular way of combining two of the most important domains in constraint logic programming: arithmetic and Boolean algebra.

In this paper we define a new constraint logic programming language \( CLP(PB) \) for logic programming with pseudo-Boolean constraints. The language is an instance of the general constraint logic programming language scheme \( CLP(X) \) and inherits all the typical semantic properties. We show that any pseudo-Boolean constraint has a most general solution and give variable elimination algorithms for pseudo-Boolean unification and unconstrained pseudo-Boolean optimization. Both algorithms subsume the well-known Boolean unification algorithm of Büttner and Simonis.

Keywords

Constraint Logic Programming, Operations Research, Pseudo-Boolean Unification, Pseudo-Boolean Optimization, 0-1 Programming
## Contents

1 Introduction .................................................. 2

2 Boolean and Pseudo-Boolean Functions ................. 3

3 Logic Programming with Pseudo-Boolean Constraints .................. 7

4 Solving Pseudo-Boolean Constraints ................. 8
   4.1 A Variable Elimination Algorithm for Pseudo-Boolean Unification .......... 8
   4.2 Solving Arbitrary Pseudo-Boolean Constraints .................. 11

5 Pseudo-Boolean Optimization .................. 11
   5.1 The Basic Algorithm for Pseudo-Boolean Optimization ............... 12
   5.2 The Special Version of the Basic Algorithm .................. 16

6 Example .................................................. 17

7 Conclusion and Further Research .................. 18

A Tree-Width of a Graph .............................. 20
1 Introduction

Boolean constraints play an important role in various constraint logic programming languages [Col87, DvHS'88, ASS'88]. In this paper we consider pseudo-Boolean constraints, that is equations and inequalities between pseudo-Boolean functions. A pseudo-Boolean function is an integer-valued function $f : \{0, 1\}^n \rightarrow \mathbb{Z}$ and thus generalizes the notion of a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$.

Pseudo-Boolean functions have been studied for a long time [HR68]. There exist many interesting methods and results on the solution of pseudo-Boolean constraints that have been developed mainly in the area of operations research. It seems very natural to apply these techniques in the context of constraint logic programming. A constraint logic programming language can be considerably enhanced by allowing pseudo-Boolean constraints. Many problems from various application areas can be expressed very naturally using pseudo-Boolean functions.

Consider for example the following knapsack problem. Suppose there is a vessel with capacity $w$ and goods $g_i$ with weight $w_i$ and value $v_i$ for $i = 1, \ldots, 12$. We introduce a Boolean variable $X_i; i = 1; \ldots; 12$; which indicates whether or not the good $g_i$ is loaded on the vessel. The possible cargos not exceeding the capacity of the vessel can be determined by the following Horn clause involving a pseudo-Boolean constraint

$$\text{cargo}(X_1, \ldots, X_{12}) : -
\text{w}_1X_1 + \ldots + \text{w}_{12}X_{12} \leq w.$$  

If we want to find the most valuable cargo, this can be done using a metapredicate max/2 that maximizes a pseudo-Boolean function subject to some pseudo-Boolean constraint.

$$\text{most-valuable-cargo}(X_1, \ldots, X_{12}) : -
\text{max}(v_1X_1 + \ldots + v_{12}X_{12}, \text{cargo}(X_1, \ldots, X_{12})).$$  

An interesting connection between logic and pseudo-Boolean optimization is that inference problems in propositional logic can be translated into linear pseudo-Boolean optimization problems. For example, suppose we want to know whether $X_2$ is implied by the set of clauses

$$X_1 \lor X_2 \lor X_3,$$
$$\overline{X_1} \lor X_2 \lor \overline{X_3},$$
$$\overline{X_1} \lor X_3 \lor X_4,$$
$$X_1 \lor \overline{X_3} \lor \overline{X_4}.$$  

In order to solve this problem we may minimize $X_2$ subject to

$$+X_1 +X_2 +X_3 \geq 1,$$
$$-X_1 +X_2 -X_4 \geq 1 - 2,$$
$$-X_1 +X_3 \geq 1 - 1,$$
$$-X_1 -X_3 +X_4 \geq 1 - 2,$$
$$+X_1 -X_3 \geq 1 - 1.$$  

The logical implication holds if and only if the minimum of $X_2$ is equal to 1.

There are many remarkable parallels between theorem proving and mathematical programming. The logical concepts of resolution, extended resolution, input and unit refutation, the Davis-Putnam-Procedure, and drawing of inferences pertinent to a given topic are closely related to the
mathematical concepts of cutting planes, Chvátal’s method, elementary closure, branch and bound, and projection of a polytope, respectively (see [Hoo88] for a survey).

More generally, pseudo-Boolean constraints can be seen as a particular way of combining two of the most important domains in constraint logic programming: arithmetic and Boolean algebra.

So far we considered only linear pseudo-Boolean functions. In many applications however one has to deal with non-linear functions and non-linear constraints. For example, suppose that we want to model the interaction of \( n \) objects \( ob_1, ob_2, \ldots, ob_n \), each of which can be chosen \( (X_i = 1) \) or not \( (X_i = 0) \). For any pair of objects \( (ob_i, ob_j) \) let \( a_{ij} \) measure the interaction between \( ob_i \) and \( ob_j \). If the global interaction is given by the sum of the interactions between all pairs of chosen objects, then it can be written as a quadratic pseudo-Boolean function \( \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} X_i X_j \) of the \( n \) variables \( X_1, \ldots, X_n \). Maximizing or minimizing the global interaction means maximizing or minimizing this pseudo-Boolean function.

Typical applications of non-linear pseudo-Boolean functions include sequencing problems, timetable scheduling, coding theory, plant location [HR68], inter-city traffic [Rhy70], kinetic energy in spin-glass models [KGV83], or supply support of space stations [FGGB66].

The organization of the paper is as follows: We start in Section 2 with some basic facts about Boolean and pseudo-Boolean functions. In Section 3 we define a new constraint logic programming language \( CLP(PB) \) for logic programming with pseudo-Boolean constraints which is an instance of the general constraint logic programming language scheme \( CLP(X) \). In Section 4 we present some of the ideas that can be used in order to solve pseudo-Boolean constraints. We show that any system of pseudo-Boolean constraints has a most general solution and give a variable elimination algorithm for computing most general pseudo-Boolean unifiers. In Section 5 we consider the problem of optimizing pseudo-Boolean functions. We present a variable elimination algorithm for pseudo-Boolean optimization and discuss its relationship to the well-known Boolean unification algorithm of Büttner and Simonis [BS87]. In Section 6 we give a typical example for the application of our concepts. Finally, Section 7 contains the conclusion and a discussion of further work.

2 Boolean and Pseudo-Boolean Functions

We begin with some basic facts about Boolean and pseudo-Boolean functions. At some places we will use the terminology of equational logic (see [HO80] for additional information).

**Definition 1** A **Boolean function** is a mapping \( f : \{0, 1\}^n \to \{0, 1\} \).

A **pseudo-Boolean function** is a mapping \( g : \{0, 1\}^n \to \mathbb{Z} \) where \( \mathbb{Z} \) denotes the ring of integer numbers.

If \( g : \{0, 1\}^n \to \mathbb{Z} \) is a pseudo-Boolean function, then the Boolean function \( g^\# : \{0, 1\}^n \to \{0, 1\} \) defined by

\[
g^\#(X_1, \ldots, X_n) = \begin{cases} 
1 & \text{if } g(X_1, \ldots, X_n) \neq 0 \\
0 & \text{if } g(X_1, \ldots, X_n) = 0
\end{cases}
\]

is called the **reduct** of \( g \).

Next we introduce a signature that allows to express Boolean and pseudo-Boolean functions as terms.
**Definition 2** Consider the order-sorted signature with two sort symbols \( \text{bool} \subseteq \text{integer} \) and the sets of function symbols

\[
\begin{align*}
F_{\text{bool}} & = \{0, 1 : \text{bool}, \neg : \text{bool} \to \text{bool}, \oplus, \land, \lor : \text{bool} \times \text{bool} \to \text{bool}\} \\
F_{\text{psbool}} & = \{0, 1 : \text{integer}, +, \times : \text{integer} \times \text{integer} \to \text{integer}, - : \text{integer} \to \text{integer}\}
\end{align*}
\]

Let

\[
\begin{align*}
V_{\text{bool}} & = \{X, Y, Z, X_1, X_2, \ldots\} \\
V_{\text{integer}} & = \{x, y, z, x_1, x_2, \ldots\}
\end{align*}
\]

denote countably infinite sets of variables of sort \( \text{bool} \) and \( \text{integer} \). The terms in the term algebra \( T(F_{\text{bool}}, V_{\text{bool}}) \) are called Boolean terms and those in \( T(F_{\text{psbool}}, V_{\text{bool}}) \) pseudo-Boolean terms.

By interpreting \( \cdot \) as negation, \( \oplus \) as sum modulo 2, \( \land \) as conjunction, and \( \lor \) as disjunction, any Boolean term \( t \in T(F_{\text{bool}}, V_{\text{bool}}) \) containing \( n \) variables generates a \( n \)-ary Boolean function \( b_t : \{0, 1\}^n \to \{0, 1\} \). Similarly, by interpreting the function symbols \( +, \times, - \) as the usual arithmetical operations on integer numbers any pseudo-Boolean term \( t \in T(F_{\text{psbool}}, V_{\text{bool}}) \) containing \( n \) variables generates a \( n \)-ary pseudo-Boolean function \( b_t : \{0, 1\}^n \to \mathbb{Z} \).

For example, the Boolean term \( X \oplus Y \oplus X \land Y \) generates the Boolean function \( f : \{0, 1\}^2 \to \{0, 1\}, f(0, 0) = 0, f(0, 1) = f(1, 0) = f(1, 1) = 1 \). The pseudo-Boolean term \( X+Y+X\times Y \) generates the pseudo-Boolean function \( g : \{0, 1\}^2 \to \mathbb{Z}, g(0, 0) = 0, g(0, 1) = g(1, 0) = 1, g(1, 1) = 3 \).

Conversely, any Boolean or pseudo-Boolean function has a canonical term representation, the polynomial normal form, which is introduced in the next proposition.

**Proposition 3**

1. For any Boolean function \( f : \{0, 1\}^n \to \{0, 1\} \) there is up to associativity and commutativity of \( \oplus, \land \) a unique Boolean term of the form

\[
\bigoplus_{I \in \Omega} \bigwedge_{i \in I} X_i, \tag{1}
\]

with a collection \( \Omega \) of subsets of \( \{1, \ldots, n\} \), that generates \( f \).

2. For any pseudo-Boolean function \( g : \{0, 1\}^n \to \mathbb{Z} \) there is up to associativity and commutativity of \( +, \times \) a unique pseudo-Boolean term of the form

\[
\sum_{I \in \Omega} c_I \prod_{i \in I} X_i, \tag{2}
\]

with a collection \( \Omega \) of subsets of \( \{1, \ldots, n\} \) and coefficients \( c_I \in \mathbb{Z} \setminus \{0\} \), that generates \( g \).

**Proof:** See [HR68] \( \square \)

**Example 4** The function \( g : \{0, 1\}^3 \to \mathbb{Z} \) defined by the table
\[
g(X, Y, Z) = -5 + 2X + 5Y + 5Z - 2XY + 6XZ - 5YZ
\]

has the polynomial normal form

\[
g(X, Y, Z) = -5 + 2X + 5Y + 5Z - 2XY + 6XZ - 5YZ
\]

Note that there exist also other representations, for example

\[
g(X, Y, Z) = 2XY + 6XZ - 5YZ,
\]

where \( \overline{X} \) is an abbreviation for \( 1 - X \).

In order to obtain a syntactic characterization of those terms that generate the same function we need equational theories for Boolean and pseudo-Boolean terms. An equational theory is a congruence relation \( \equiv_E \) on the term algebra which is generated by a set of equations \( E \). For two terms \( s, t \) we have \( s \equiv_E t \) if and only if \( s \) and \( t \) can be proven equal using the equations in \( E \) [HO80].

**Definition 5** 1. The *equational theory* \( \equiv_B \) of Boolean terms is generated by the set of equations

\[
B = \{ X \oplus (Y \oplus Z) \doteq (X \oplus Y) \oplus Z, \\
X \oplus 0 \doteq X, \\
X \oplus X \doteq 0, \\
X \oplus Y \doteq Y \oplus X, \\
X \land (Y \land Z) \doteq (X \land Y) \land Z, \\
X \land 1 \doteq X, \\
X \land Y \doteq Y \land X, \\
X \land (Y \oplus Z) \doteq (X \land Y) \oplus (X \land Z), \\
X \land X \doteq X, \\
\overline{X} \doteq X \oplus 1, \\
X \lor Y \doteq (X \oplus Y) \lor X \land Y \}
\]

2. The *equational theory* \( \equiv_{PB} \) of pseudo-Boolean terms is generated by the set of equations

\[
PB = \{ x + (y + z) \doteq (x + y) + z, \\
x + 0 \doteq x,
\]

5
For any Boolean term

\[ x + (-x) \equiv 0, \]
\[ x + y \equiv y + x, \]
\[ x \ast (y \ast z) \equiv (x \ast y) \ast z, \]
\[ x \ast 1 \equiv x, \]
\[ x \ast y \equiv y \ast x, \]
\[ x \ast (y + z) \equiv (x \ast y) + (x \ast z), \]
\[ X \ast X \equiv X \}

The next proposition shows that two terms denote the same function iff they are congruent modulo the equational theory \( \equiv_{PB} \) or \( \equiv_B \) respectively.

**Proposition 6**

1. Two Boolean terms \( s, t \in T(F_{bool}, V_{bool}) \) generate the same Boolean function iff they are congruent modulo the theory \( B \), that is \( \hat{s} = \hat{t} \) iff \( s \equiv_B t \).

2. Two pseudo-Boolean terms \( s, t \in T(F_{psbool}, V_{bool}) \) generate the same pseudo-Boolean function iff they are congruent modulo the theory \( PB \), that is \( \hat{s} = \hat{t} \) iff \( s \equiv_{PB} t \).

**Proof:** We prove only the second part of the theorem. The first part is completely analogous.

“\( \leftarrow \rightleftharpoons \):” If \( s \equiv_{PB} t \), then by the well-known theorem of Birkhoff [HO80] the equation \( s \equiv t \) holds in all models of the set of equations \( PB \), in particular in the algebra \( \{0, 1\}, \mathcal{Z} \). This means that for any variable assignment \( \sigma : V_{bool} \rightarrow \{0, 1\} \) the terms \( s \) and \( t \) have the same value. This is equivalent to say that the pseudo-Boolean functions generated by \( s \) and \( t \) are the same.

“\( \rightarrow \rightleftharpoons \):” Using the equations in \( PB \) we may transform the pseudo-Boolean terms \( s, t \) to pseudo-Boolean terms \( s_{\downarrow} \equiv_{PB} s \) and \( t_{\downarrow} \equiv_{PB} t \) such that \( s_{\downarrow} \) and \( t_{\downarrow} \) are of the form (2). From “\( \leftarrow \rightleftharpoons \)” we deduce that \( \hat{s} = \hat{s}_{\downarrow} \) and \( \hat{t} = \hat{t}_{\downarrow} \). From the preceding proposition and the assumption \( \hat{s} = \hat{t} \) we know that \( s_{\downarrow} \) and \( t_{\downarrow} \) are the same up to associativity and commutativity of \( +, \ast \). This implies \( s_{\downarrow} \equiv_{PB} t_{\downarrow} \) and consequently \( s \equiv_{PB} t \). \( \square \)

Boolean functions can be described by both Boolean and pseudo-Boolean terms. A canonical transformation between the two representations is given in the next definition.

**Definition 7** The pseudo-Boolean form \( t^P \in T(F_{psbool}, V_{bool}) \) of a term \( t \in T(F_{bool} \cup F_{psbool}, V_{bool}) \) is obtained by normalizing \( t \) using the canonical term rewrite system

\[
P = \{ X \oplus Y \rightarrow X + Y - (X \ast Y + X \ast Y), \]
\[ X \wedge Y \rightarrow X \ast Y, \]
\[ \overline{X} \rightarrow 1 - X, \]
\[ X \lor Y \rightarrow X + Y - X \ast Y \} \]

**Lemma 8** For any Boolean term \( t \in T(F_{bool}, V_{bool}) \) the functions generated by \( t \) and \( t^P \) agree, that is \( \hat{t} = \hat{t^P} \).

**Proof:** The equations \( X \oplus Y \equiv X + Y - (X \ast Y + X \ast Y), X \wedge Y \equiv X \ast Y, \overline{X} \equiv 1 - X, X \lor Y \equiv X + Y - X \ast Y \) hold in the algebra \( \{0, 1\}, \mathcal{Z} \). This implies \( \{0, 1\}, \mathcal{Z} \models t \models t^P \) and we get \( \hat{t} = \hat{t^P} \). \( \square \)
Lemma 9 Let $s, t \in \mathcal{T}(F_{\text{bool}}, V_{\text{bool}})$ be two Boolean terms. Then $s \equiv_B t$ iff $s^P \equiv_P t^P$.

Proof: By proposition 6 we have $s \equiv_B t$ iff $\hat{s} = \hat{t}$. By the preceding lemma $\hat{s} = s^P$ and $\hat{t} = t^P$. Again by proposition 6, $s^P \equiv_P t^P$ iff $s^P = t^P$. Altogether this yields $s \equiv_B t$ iff $\hat{s} = \hat{t}$ iff $s^P = t^P$ iff $s^P \equiv_P t^P$. \hfill \Box

We close this section by an easy lemma that we will need several times in the sequel.

Lemma 10 Suppose $f, g : \{0, 1\}^n \rightarrow \mathbb{Z}$ are pseudo-Boolean functions and $u : \{0, 1\}^n \rightarrow \{0, 1\}$ is a Boolean function. Then $f \ast u + g \ast \overline{u} = 0$ implies $f \ast u = g \ast \overline{u} = 0$ and $f \ast g = 0$.

Proof: Assume $f \ast u + g \ast \overline{u} = 0$. Multiplying this equation with $u$ yields $0 = f \ast u \ast u + g \ast \overline{u} \ast u = f \ast u$. In the same way, multiplication with $\overline{u}$ yields $0 = f \ast u \ast \overline{u} + g \ast \overline{u} \ast \overline{u} = g \ast \overline{u}$. Finally, $f \ast g = f \ast g \ast (u + \overline{u}) = f \ast g \ast u + f \ast g \ast \overline{u} = 0 + 0 = 0$. \hfill \Box

3 Logic Programming with Pseudo-Boolean Constraints

We now introduce a constraint logic programming language $\text{CLP}(PB)$ over an algebraic structure $PB$ that allows us to handle equations and inequalities between pseudo-Boolean functions. The language is an instance of the constraint logic programming language scheme $\text{CLP}(X)$ of [JL86, JL87].

The algebraic structure in question is the order-sorted algebra $PB$ over the signature $\Sigma = (S, F, P)$ with

$$S = \{ \text{bool} \subseteq \text{integer} \subseteq \text{tree} \}$$

$$F = F_{\text{bool}} \cup F_{\text{psbool}} \cup \{ f_i : \text{tree}^{n_i} \rightarrow \text{tree} \mid i = 1, \ldots, k, k \geq 0, n_i \geq 0 \}$$

$$P = \{ \preceq, \leq, <, \geq, > : \text{integer} \times \text{integer} \rightarrow \text{integer} \}$$

where the interpretation of the sort, function and predicate symbols is the usual one.

$V = V_{\text{bool}} \cup V_{\text{integer}} \cup V_{\text{tree}}$ denotes a collection of countably infinite sets of variables of the different sorts.

An atomic pseudo-Boolean constraint is of the form

$$s \preceq t, u \leq v, u < v, u \geq v, \text{ or } u > v$$

with terms $s, t \in \mathcal{T}(F, V)$ and pseudo-Boolean terms $u, v \in \mathcal{T}(F_{\text{psbool}}, V_{\text{bool}})$. A pseudo-Boolean constraint is a possibly empty finite set of atomic constraints.

An atom is of the form $p(t_1, \ldots, t_n)$ where $p$ is a predicate symbol distinct from $\preceq, \leq, <, \geq, >$ and $t_i \in \mathcal{T}(F, V), i = 1, \ldots, n, n \geq 0$, are terms.

A $\text{CLP}(PB)$ constraint logic program is defined over a signature $\Sigma^* = (S, F, P^*)$ where $P^* \cap P = \emptyset$. It consists of a finite set of constraint rules each being of the form

$$H \vdash c \mid B_1, \ldots, B_n$$

where $c$ is a possibly empty pseudo-Boolean constraint and $H, B_i, i = 1, \ldots, n, n \geq 0$, are atoms over $\Sigma^*$.  

7
A CLP(PB) goal is of the form

\[ ? - c \| B_1, \ldots, B_n \]

where again \( c \) is a possibly empty pseudo-Boolean constraint and \( B_i, i = 1, \ldots, n, n \geq 0 \), are atoms over \( \Sigma^* \).

The structure \( PB \) is solution compact because it has no limit elements. From the general results of [JL86, JL87] we can derive for the language CLP(PB):

- the existence of a canonical domain of computation
- the existence of a least and greatest model semantics
- the existence of a least and greatest fixpoint semantics
- soundness and completeness results for successful derivations of the underlying implementation model
- soundness and completeness results for finitely failed derivations of the underlying implementation model
- soundness and completeness results for negation as failure

4 Solving Pseudo-Boolean Constraints

In this section we describe how we may solve arbitrary pseudo-Boolean constraints. A detailed account of this subject is beyond the scope of this paper. We will restrict ourselves mainly to the case of equality constraints. In addition, we show how arbitrary pseudo-Boolean constraints can be reduced to pseudo-Boolean equations. However, such a reduction is mainly of theoretical interest. In practice, it will be more efficient to handle non-equality constraints by a specific solving procedure.

4.1 A Variable Elimination Algorithm for Pseudo-Boolean Unification

**Definition 11** A pseudo-Boolean unification problem is a system of equations

\[ S : s_1 = t_1, \ldots, s_n = t_n \]

with pseudo-Boolean terms \( s_i, t_i \in \mathcal{T}(F_{psbool}, V_{bool}), i = 1, \ldots, n \). A pseudo-Boolean unifier of \( S \) is a substitution \( \tau : V_{bool} \rightarrow \mathcal{T}(F_{bool}, V_{bool}) \) such that

\[ \tau(s_i)^P \equiv_{PB} \tau(t_i)^P, \text{ for all } i = 1, \ldots, n. \]

A most general pseudo-Boolean unifier of \( S \) is a substitution \( \sigma : V_{bool} \rightarrow \mathcal{T}(F_{bool}, V_{bool}) \) such that for all pseudo-Boolean unifiers \( \tau \) of \( S \) there is a substitution \( \lambda : V_{bool} \rightarrow \mathcal{T}(F_{bool}, V_{bool}) \) with

\[ \lambda(\sigma(X)) \equiv_B \tau(X), \text{ for all } X \in V_{bool}. \]
Now we want to give a constructive proof that, similar to syntactic unification in classical logic programming, any two unifiable pseudo-Boolean terms have a most general pseudo-Boolean unifier. In the terminology of unification theory this means that pseudo-Boolean terms form a unitary theory. The next theorem is a generalization of the well-known Boolean unification method of [BS87] to the pseudo-Boolean case. For the proof we use previous results by P. Hammer [Ham64a].

**Theorem 12** Pseudo-Boolean unification is unitary, that is any two pseudo-Boolean terms are either not unifiable modulo PB or they have a most general pseudo-Boolean unifier.

**Proof:** It is sufficient to consider the pseudo-Boolean unification problem \( u \ni 0 \), with a pseudo-Boolean term \( u \in T(F_{\text{psbool}}, V_{\text{bool}}) \). Let \( \text{Var}(u) = \{X_1, \ldots, X_{n_u}\} \) be the set of variables occurring in \( u \). The proof uses induction on \( n_u \) and provides a decision procedure for the unifiability of \( u \) and 0 as well as an algorithm for computing a most general unifier.

To simplify our notation we will identify throughout the proof a Boolean or pseudo-Boolean term \( t \) with the corresponding function \( \hat{t} \). In particular, we will write \( s = t \) instead of \( \hat{s} = \hat{t} \). This is justified by the considerations in Section 2.

First let \( n_u = 0 \). If \( u = 0 \) then the empty substitution is a most general pseudo-Boolean unifier, and if \( u \neq 0 \) then the equation is unsolvable.

Suppose now that the theorem is true for pseudo-Boolean terms which contain at most \( n \) variables, \( n \geq 0 \), and let \( u \) be a pseudo-Boolean term with \( n + 1 \) variables. Select a variable \( X_{n+1} \) in \( u \) and write \( u \) in the form

\[
u = v \cdot X_{n+1} + w \cdot \overline{X_{n+1}}
\]

where \( \{X_1, \ldots, X_n\}, n \geq 0 \), are the only variables occurring in the pseudo-Boolean terms \( v, w \).

Let \( \tau : V_{\text{bool}} \rightarrow T(F_{\text{psbool}}, V_{\text{bool}}) \) be a pseudo-Boolean unifier of \( u \) and 0. Then we have

\[
\tau(v) \cdot \tau(X_{n+1}) + \tau(w) \cdot \overline{\tau(X_{n+1})} = 0.
\]

From Lemma 10 we deduce that \( \tau(v) \cdot \tau(w) = \tau(v \cdot w) = 0 \) or that \( \tau \) is a pseudo-Boolean unifier of \( v \cdot w \) and 0.

Conversely, if \( \rho : V_{\text{bool}} \rightarrow T(F_{\text{psbool}}, V_{\text{bool}}) \) is a unifier of \( v \cdot w \) and 0, then

\[
\sigma(X_i) \overset{\text{def}}{=} \rho(X_i), \text{ for } i = 1, \ldots, n
\]

\[
\sigma(X_{n+1}) \overset{\text{def}}{=} \frac{\rho(v \overline{Y}) \land Y + \rho(w \overline{Y}) \land \overline{Y}}{= \rho(v \overline{Y}) \ast Y + \rho(w \overline{Y}) \ast \overline{Y}},
\]

with a new variable \( Y \in V_{\text{bool}} \), defines a pseudo-Boolean unifier of \( u \) and 0. In fact, we have

\[
\sigma(u) = \sigma(v) \ast \sigma(X_{n+1}) + \sigma(w) \ast \overline{\sigma(X_{n+1})}
\]

\[
= \rho(v) \ast \overline{\rho(v \overline{Y}) \ast Y + \rho(w \overline{Y}) \ast \overline{Y}} + \rho(w) \ast \overline{\rho(v \overline{Y}) \ast Y + \rho(w \overline{Y}) \ast \overline{Y}}
\]

\[
= \rho(v) \ast \overline{\rho(v \overline{Y}) \ast Y + \rho(w \overline{Y}) \ast \overline{Y}} + \rho(w) \ast (1 - \rho(v \overline{Y}) \ast Y - \rho(w \overline{Y}) \ast \overline{Y})
\]

\[
= 0 + 0 + \rho(w) - \rho(w) \ast \overline{\rho(v \overline{Y}) \ast Y - \rho(w) \ast \rho(w \overline{Y}) \ast \overline{Y}}
\]

\[
= \rho(w) - \rho(w) \ast (1 - \rho(w \overline{Y})) \ast Y - \rho(w) \ast (1 - Y)
\]

\[
= \rho(w) - \rho(w) \ast Y + \rho(w) \ast \rho(v \overline{Y}) \ast Y - \rho(w) + \rho(w) \ast Y = 0
\]

Here, we have used the identities \( \rho(v) \ast \rho(v \overline{Y}) = 0, 0 = \rho(v) \ast \rho(w) = \rho(v \overline{Y}) \ast \rho(w) = \rho(v) \ast \rho(w \overline{Y}) = \rho(v \overline{Y}) \ast \rho(w \overline{Y}), \) and \( \rho(w) \ast \rho(w \overline{Y}) = \rho(w) \).
Solved
\[ \{ u \equiv 0 \} \]
\[ \text{true} \]
if \( u \equiv_{PB} 0 \)

Unsolvable
\[ \{ u \equiv 0 \} \]
\[ \text{fail} \]
if \( u \not\equiv_{PB} 0 \) and \( \text{Var}(u) = \emptyset \)

Variable Elimination
\[ \{ u \equiv 0 \} \]
\[ \{ v \ast w \equiv 0 \} \cup \{ X \equiv \bar{v} \ast Y + \bar{w} \ast \bar{Y} \} \]
if \( u \not\equiv_{PB} 0 \),
\( u \equiv_{PB} v \ast X + w \ast X \),
\( X \not\in \text{Var}(v, w) \),
and \( Y \) is a new variable

Figure 1: Variable Elimination for Pseudo-Boolean Unification

It remains to show that if \( \rho \) is a most general pseudo-Boolean unifier of \( v \ast w \equiv 0 \), then \( \sigma \) is a most general pseudo-Boolean unifier of \( u \equiv 0 \). Let \( \tau \) be an arbitrary unifier of \( u \) and 0. The restriction \( \tau' \) of \( \tau \) to the variables \( \{X_1, \ldots, X_n\} \) unifies \( v \ast w \) and 0. Consequently, there exists a substitution \( \lambda' \) such that
\[ \tau'(X_i) = \lambda'(\rho(X_i)) \], for all \( i = 1, \ldots, n \).

We define a substitution \( \lambda \) by
\[ \lambda(X) \overset{\text{def}}{=} \lambda'(X), \text{ for } X \in \bigcup_{i=1,\ldots,n} \text{Var}(\rho(X_i)) \]
\[ \lambda(Y) \overset{\text{def}}{=} \tau(X_{n+1}). \]

Now we get
\[ \lambda(\sigma(X_{n+1})) = \lambda'(\rho(v^{\bar{x}})) \ast \lambda(Y) + \lambda'(\rho(w^{\bar{x}})) \ast \lambda(Y) \]
\[ = \lambda'(\rho(v^{\bar{x}})) \ast \tau(X_{n+1}) + \lambda'(\rho(w^{\bar{x}})) \ast \tau(X_{n+1}) \]
\[ = \tau(v^{\bar{x}}) \ast \tau(X_{n+1}) + \tau(w^{\bar{x}}) \ast \tau(X_{n+1}) \]
\[ = (1 - \tau(v^{\bar{x}})) \ast \tau(X_{n+1}) + \tau(w^{\bar{x}}) \ast \tau(X_{n+1}) \]
\[ = \tau(X_{n+1}) - \tau(v^{\bar{x}}) \ast \tau(X_{n+1}) + \tau(w^{\bar{x}}) \ast \tau(X_{n+1}) \]
\[ = \tau(X_{n+1}) - 0 = \tau(X_{n+1}) \]

The last equality holds, because \( \tau \) is a unifier of \( u \) and 0. This means \( \tau(v) \ast \tau(X_{n+1}) + \tau(w) \ast \tau(X_{n+1}) = 0 \) which by Lemma 10 implies \( \tau(v^{\bar{x}}) \ast \tau(X_{n+1}) + \tau(w^{\bar{x}}) \ast \tau(X_{n+1}) = 0 \). Altogether we have shown that \( \sigma \) is a most general pseudo-Boolean unifier of \( u \) and 0. \( \square \)
From the proof of the preceding theorem we derive a variable elimination algorithm for pseudo-Boolean unification which subsumes the Boolean unification algorithm [BS87]. Given a pseudo-Boolean term $u$, it computes a most general pseudo-Boolean unifier of $u$ and 0 if it exists and fails otherwise (see Figure 1).

### 4.2 Solving Arbitrary Pseudo-Boolean Constraints

We show in this section how arbitrary pseudo-Boolean constraints can be reduced to equality constraints.

**Lemma 13** The problem of solving a pseudo-Boolean inequality is reducible to that of solving a pseudo-Boolean equation.

**Proof:** It is enough to consider a pseudo-Boolean inequality of the form

$$g(X_1, \ldots, X_n) \leq 0. \tag{3}$$

The following proof was first given in [Ham64b]. Let $g(X_1, \ldots, X_n) = \sum_{I \in \Omega} c_I \prod_{i \in I} X_i$ be a polynomial representation of $g$ and let $A \overset{\text{def}}{=} \sum_{c_I < 0} c_I$. Then $A$ is a lower bound of the pseudo-Boolean function $g$. Let $-A = a_0 2^0 + \cdots + a_k 2^k, k \geq 0$, be the binary representation of $-A \geq 0$. The inequality (3) is equivalent to the equation

$$g(X_1, \ldots, X_n) + y = 0 \tag{4}$$

with the additional restriction $y \geq 0$. Therefore we may write $y = Y_0 2^0 + \cdots + Y_l 2^l, l \geq 0$, and since $y = -g(X_1, \ldots, X_n) \leq -A$ we get $l \leq k$. It follows that (3) and (4) are equivalent to the pseudo-Boolean equation

$$g(X_1, \ldots, X_n) + Y_0 2^0 + \cdots + Y_k 2^k = 0 \tag{5}$$

where in the case $k > l$ we have $Y_{l+1} = \cdots = Y_k = 0$.

**Lemma 14** The problem of solving a system of pseudo-Boolean equations is reducible to that of solving a single pseudo-Boolean equation.

**Proof:** The system $s_1 \overset{\text{def}}{=} t_1, \ldots, s_n \overset{\text{def}}{=} t_n, n \geq 1$, is equivalent to the equation $\sum_{i=1}^n (s_i - t_i)^2 = 0$.

The two lemmas together yield

**Proposition 15** The problem of solving an arbitrary pseudo-Boolean constraint is reducible to that of solving a single pseudo-Boolean equation.

### 5 Pseudo-Boolean Optimization

In the last section we have sketched a solution procedure for pseudo-Boolean constraints. In practical applications, one is often not interested in all solutions of a pseudo-Boolean constraint but only in those solutions which are optimal with respect to some objective function, which itself is a pseudo-Boolean function.
This leads to the problem of pseudo-Boolean optimization also known as nonlinear 0-1 programming.

Maximize the pseudo-Boolean function \( g(X_1, \ldots, X_n) \) subject to the pseudo-Boolean constraint \( f(X_1, \ldots, X_n) = 0 \).

A constrained pseudo-Boolean optimization problem can be transformed to an unconstrained pseudo-Boolean optimization problem by first computing a most general solution
\[
X_1 = t_1(Y_1, \ldots, Y_m), \ldots, X_n = t_n(Y_1, \ldots, Y_m)
\]
of the constraint \( f(X_1, \ldots, X_n) = 0 \) and then maximizing the pseudo-Boolean function
\[
h(Y_1, \ldots, Y_m) \overset{\text{def}}{=} g(t_1(Y_1, \ldots, Y_m), \ldots, t_n(Y_1, \ldots, Y_m)).
\]

The unconstrained quadratic pseudo-Boolean maximization problem \( z = \max(g) \) with a quadratic pseudo-Boolean function \( g : \{0, 1\}^n \rightarrow \mathbb{Z} \) is NP-complete. As a matter of fact, most of the well-known NP-complete problems (for example the minimum cut problem, balancing a signed graph, maximum 2-satisfiability) can be very naturally formulated as unconstrained quadratic pseudo-Boolean optimization problems. The same is true for broad classes of constrained pseudo-Boolean optimization problems, such as the maximization of a linear function subject to a quadratic Boolean equation or the maximization of a quadratic function subject to a system of linear equations [HS86].

5.1 The Basic Algorithm for Pseudo-Boolean Optimization

There is an extensive literature on the solution of pseudo-Boolean optimization problems (see the survey [HS86]) that we cannot cover in this paper. Because of its close relationship to the variable elimination method for Boolean unification we present here a variant of the so-called Basic Algorithm for pseudo-Boolean optimization. This algorithm was originally proposed in [HRR63a, HRR63b] and allows to maximize an arbitrary nonlinear pseudo-Boolean function \( f : \{0, 1\}^n \rightarrow \mathbb{Z} \).

First we define the notion of a most-general maximizer of a pseudo-Boolean function \( f \).

**Definition 16** Let \( f : \{0, 1\}^n \rightarrow \mathbb{Z} \) be a pseudo-Boolean function with the maximum value \( z \in \mathbb{Z} \). Let \( t_f \in T(F_{psbool}, V_{bool}) \) be a pseudo-Boolean term representing \( f \). A (most general) maximizer of \( f \) (resp. \( t_f \)) is a (most general) pseudo-Boolean unifier of the pseudo-Boolean equation \( t \overset{\text{uf}}{=} z \).

From Theorem 12 we can conclude that any pseudo-Boolean function \( f : \{0, 1\}^n \rightarrow \mathbb{Z} \) has a most general maximizer \( \sigma_f \) which subsumes any other maximizer \( \tau \) of \( f \). However, we cannot compute \( \sigma_f \) by the variable elimination algorithm for pseudo-Boolean unification, because we do not know a priori the maximum value of \( f \). The Basic Algorithm for pseudo-Boolean optimization is a variable elimination method which computes simultaneously the maximum value \( z \) of \( f \) and a most general pseudo-Boolean unifier of the equation \( f \overset{\text{uf}}{=} z \).

In order to formulate the algorithm we need the following definitions.

**Definition 17** Let \( g : \{0, 1\}^n \rightarrow \mathbb{Z} \) be a pseudo-Boolean function. The pseudo-Boolean function \( g^+ : \{0, 1\}^n \rightarrow \mathbb{Z} \) is defined by
\[
g^+(X_1, \ldots, X_n) \overset{\text{def}}{=} \begin{cases} g(X_1, \ldots, X_n), & \text{if } g(X_1, \ldots, X_n) > 0 \\ 0, & \text{if } g(X_1, \ldots, X_n) \leq 0. \end{cases}
\]
Definition 18 Let \( g : \{0,1\}^n \rightarrow \mathbb{Z} \) be a pseudo-Boolean function. The Boolean functions \( g^- : \{0,1\}^n \rightarrow \{0,1\} \) and \( g^+ : \{0,1\}^n \rightarrow \{0,1\} \) are defined by

\[
g^-(X_1, \ldots, X_n) \overset{\text{def}}{=} \begin{cases} 1, & \text{if } g(X_1, \ldots, X_n) = 0 \\ 0, & \text{if } g(X_1, \ldots, X_n) \neq 0 \end{cases}
\]

and

\[
g^+(X_1, \ldots, X_n) \overset{\text{def}}{=} \begin{cases} 1, & \text{if } g(X_1, \ldots, X_n) > 0 \\ 0, & \text{if } g(X_1, \ldots, X_n) \leq 0 \end{cases}
\]

Now we can describe the Basic Algorithm by a set of inference rules (see Figure 2).

The correctness proof of the Basic Algorithm depends on the following two lemmas that relate a most general maximizer of a pseudo-Boolean function \( f = g * X + h \) to a most general maximizer of \( g^+ + h \).

Lemma 19 Let \( f : \{0,1\}^{n+1} \rightarrow \mathbb{Z} \) be a pseudo-Boolean function. Suppose

\[
f(X_1, \ldots, X_{n+1}) = X_{n+1} * g(X_1, \ldots, X_n) + h(X_1, \ldots, X_n)
\]

with pseudo-Boolean functions \( g, h : \{0,1\}^n \rightarrow \mathbb{Z} \). Then

\[
\text{max}(f) = \text{max}(g^+ + h).
\]

If \( \tau : \{X_1, \ldots, X_{n+1}\} \rightarrow T(F_{\text{bool}}, V_{\text{bool}}) \) is a maximizer of \( f \) then the restriction of \( \tau \) to \( \{X_1, \ldots, X_n\} \) is a maximizer of \( g^+ + h \).

Moreover, \( \tau \) satisfies the condition \( \tau(X_{n+1}) = \tau(g^-) * \tau(X_{n+1}) + \tau(g^+) \).

Proof: As before, we will not distinguish a pseudo-Boolean function \( f \) and the corresponding term \( t_f \).

From \( f = X_{n+1} * g + h \leq X_{n+1} * g^+ + h \leq g^+ + h \) we get \( \text{max}(f) \leq \text{max}(g^+ + h) \). Let \( z \in \mathbb{Z} \) denote the maximum value of \( g^+ + h \). Then there exist \( a_1, \ldots, a_n \in \{0,1\} \) such that \( (g^+ + h)(a_1, \ldots, a_n) = \)}
z. Let \( a_{n+1} \) be defined as \( g^>(a_1, \ldots, a_n) \). Then \( f(a_1, \ldots, a_{n+1}) = a_{n+1} \ast g(a_1, \ldots, a_n) + h(a_1, \ldots, a_n) = g^+(a_1, \ldots, a_n) + h(a_1, \ldots, a_n) = z \). This implies \( \max(f) \geq \max(g^+ + h) \) and altogether we get \( \max(f) = \max(g^+ + h) = z \).

Now let \( \tau : \{X_1, \ldots, X_{n+1}\} \to T(\text{F}_{\text{bool}}, \text{V}_{\text{bool}}) \) be a maximizer of \( f \).

First we show that \( \tau(X_{n+1}) = \tau(g^+) \ast \tau(X_{n+1}) + \tau(g^+) \). Otherwise there would exist a ground substitution \( \gamma : \text{V}_{\text{bool}} \to \{0, 1\} \) such that \( (\gamma \circ \tau)(X_{n+1}) \neq (\gamma \circ \tau)(g^+) \ast (\gamma \circ \tau)(X_{n+1}) + (\gamma \circ \tau)(g^+) \).

Since

\[
(\gamma \circ \tau)(g^-) \ast (\gamma \circ \tau)(X_{n+1}) + (\gamma \circ \tau)(g^+) = \begin{cases} 
1, & \text{if } (\gamma \circ \tau)(g) > 0 \\
(\gamma \circ \tau)(X_{n+1}), & \text{if } (\gamma \circ \tau)(g) = 0 \\
0, & \text{if } (\gamma \circ \tau)(g) < 0
\end{cases}
\]

we get a contradiction in the case \( (\gamma \circ \tau)(g) = 0 \). In the two remaining cases we can derive

\[
(\gamma \circ \tau)(X_{n+1}) = \begin{cases} 
0, & \text{if } (\gamma \circ \tau)(g) > 0 \\
1, & \text{if } (\gamma \circ \tau)(g) < 0
\end{cases}
\]

This is an obvious contradiction to the maximality of \( \tau \) as we show now.

Define the substitution \( \tau : \{X_1, \ldots, X_{n+1}\} \to T(\text{F}_{\text{bool}}, \text{V}_{\text{bool}}) \) by

\[
\tau(X_i) \overset{\text{def}}{=} (\gamma \circ \tau)(X_i), \text{ for } i = 1, \ldots, n
\]

\[
\tau(X_{n+1}) \overset{\text{def}}{=} \begin{cases} 
1, & \text{if } (\gamma \circ \tau)(g) > 0 \\
0, & \text{if } (\gamma \circ \tau)(g) < 0
\end{cases}
\]

Then \( \tau(f) = \tau(g \ast X_{n+1} + h) = \tau(g) \ast \tau(X_{n+1}) + \tau(h) = (\gamma \circ \tau)(g) \ast (\gamma \circ \tau)(X_{n+1}) + (\gamma \circ \tau)(g) > (\gamma \circ \tau)(g) \ast (\gamma \circ \tau)(X_{n+1}) + (\gamma \circ \tau)(h) = (\gamma \circ \tau)(f) = \gamma(z) = z \), in contradiction to the maximality of \( z \). This allows us to conclude that \( \tau(X_{n+1}) = \tau(g^-) \ast \tau(X_{n+1}) + \tau(g^+) \).

In order to show that the restriction of \( \tau \) to \( \{X_1, \ldots, X_n\} \) is maximizer of \( g^+ + h \), we must prove that \( \tau(g^+ + h) = z \). Since we know that \( \tau(f) = \tau(X_{n+1}) \ast \tau(g) + \tau(h) = z \), it is enough to show that \( \tau(X_{n+1}) \ast \tau(g) = \tau(g^+) \). But, \( \tau(X_{n+1}) \ast \tau(g) = (\tau(g^-) \ast \tau(X_{n+1}) + \tau(g^+)) \ast \tau(g) = \tau(g^-) \ast \tau(g) \ast \tau(X_{n+1}) + \tau(g^+) \ast \tau(g) = 0 + \tau(g^+) = \tau(g^+) \), and the lemma is proved.

\( \square \)

Lemma 20 Let \( f : \{0, 1\}^{n+1} \to \mathbb{Z} \) be a pseudo-Boolean function. Suppose

\[
f(X_1, \ldots, X_{n+1}) = X_{n+1} \ast g(X_1, \ldots, X_n) + h(X_1, \ldots, X_n)
\]

with pseudo-Boolean functions \( g, h : \{0, 1\}^n \to \mathbb{Z} \). Let \( \rho : \text{V}_{\text{bool}} \to T(\text{F}_{\text{bool}}, \text{V}_{\text{bool}}) \) be a most general maximizer of the pseudo-Boolean function \( g^+ + h \). Then the substitution \( \sigma : \text{V}_{\text{bool}} \to T(\text{F}_{\text{bool}}, \text{V}_{\text{bool}}) \),

\[
\sigma(X_i) \overset{\text{def}}{=} \rho(X_i), \text{ for } i = 1, \ldots, n
\]

\[
\sigma(X_{n+1}) \overset{\text{def}}{=} \rho(g^-) \land Y \lor \rho(g^+) = \rho(g^-) \land Y \lor \rho(g^+),
\]

with a new variable \( Y \in \text{V}_{\text{bool}} \), defines a most general maximizer of \( f \).

Proof: Again we do not distinguish a pseudo-Boolean function \( f \) and the corresponding term \( t_f \). Let \( z \in \mathbb{Z} \) be the maximum value of \( f \). By Lemma 19, \( z \) is also the maximum value of \( g^+ + h \). This
means that the substitution \( \rho \) is a most general pseudo-Boolean unifier of the equation \( g^+ + h \triangleq z \).

In particular, we get \( \rho(g^+) + \rho(h) = z \). It follows that

\[
\sigma(f) = \sigma(X_{n+1}) \ast \sigma(g) + \sigma(h) \\
= (\rho(g^-) \ast Y + \rho(g^+)) \ast \rho(g) + \rho(h) \\
= \rho(g^-) \ast \rho(g) \ast Y + \rho(g^+ \ast \rho(g) + \rho(h) \\
= \rho(g^- \ast g) \ast Y + \rho(g^+ \ast g) + \rho(h) \\
= \rho(0) \ast Y + \rho(g^+) + \rho(h) \\
= 0 + z = z.
\]

Here we have used the identities \( g^- \ast g = 0 \) and \( g^+ \ast g = g^+ \). This shows, that \( \sigma \) is a maximizer of \( f \).

Let \( \tau : V_{\text{bool}} \rightarrow T(F_{\text{bool}}, V_{\text{bool}}) \) be an arbitrary maximizer of \( f \), that is \( z = \tau(f) = \tau(X_{n+1}) \ast \tau(g) + \tau(h) \). By Lemma 19, \( \tau \) is also a maximizer of \( g^+ + h \). It follows that there exists a substitution \( \lambda' : V_{\text{bool}} \rightarrow T(F_{\text{bool}}, V_{\text{bool}}) \) such that \( \lambda'(\rho(X_i)) = \tau(X_i) \), for all \( i = 1, \ldots, n \).

We define a substitution \( \lambda \) by

\[
\lambda(X) \overset{\text{def}}{=} \lambda'(X), \text{ for } X \in \bigcup_{i=1,\ldots,n} \text{Var}(\rho(X_i)) \\
\lambda(Y) \overset{\text{def}}{=} \tau(X_{n+1}).
\]

Then we get \( \lambda(\sigma(X_{n+1})) = \lambda(\rho(g^-) \ast Y + \rho(g^+)) = \lambda(g^-) \ast \lambda(Y) + \lambda(g^+) = \tau(g^-) \ast \tau(X_{n+1}) \ast \tau(Y) = \tau(X_{n+1}) \).

The last equality holds because of Lemma 19.

By definition, \( \lambda(\sigma(X_i)) = \lambda'(\rho(X_i)) = \tau(X_i) \), for all \( i = 1, \ldots, n \). This shows that \( \sigma \) is even a most general maximizer of \( f \). \( \square \)

Now we are able to prove the correctness of our algorithm.

**Theorem 21** Let \( f : \{0, 1\}^n \rightarrow \mathbb{Z} \) be a pseudo-Boolean function represented by the pseudo-Boolean term \( u \in T(F_{p\text{bool}}, V_{\text{bool}}) \).

Starting with the problem \( z \triangleq \max(u) \) and applying the rules of the Basic Algorithm at most \( n + 1 \) times one obtains a solved form

\[
\{ z \triangleq c, X_1 \triangleq t_1, \ldots, X_n \triangleq t_n \}.
\]

with \( c \in \mathbb{Z}, t_i \in T(F_{p\text{bool}}, V_{\text{bool}}), \) such that \( c \) is the maximum value and \( \{ X_1 \leftarrow t_1 \circ \ldots \circ \{ X_n \leftarrow t_n \} \) is a most general maximizer of \( f \).

**Proof:** Let \( \text{Var}(u) = \{ X_1, \ldots, X_m \} \) denote the set of variables occurring in the pseudo-Boolean term \( u \). The proof is by induction on \( n_u \).

First consider the case \( n_u = 0 \). If \( u \) contains no variables, the function \( f \) is equivalent to a constant function \( c : \rightarrow \mathbb{Z} \), the rule **Constant** applies, and we get \( \max(f) = z = c \).

Now suppose \( n_u = n + 1, n \geq 0 \). We may assume that \( u \) is not congruent to a constant. Then we can choose a variable \( X_{n+1} \) in \( u \) and write \( u \) in the form

\[
u = v \ast X_{n+1} + w,
\]

15
Constant

\[
\begin{align*}
\text{if } u &\equiv_{PB} c = \text{constant} \\
\frac{z \doteq \max(u)}{z \doteq c}
\end{align*}
\]

Variable Elimination

\[
\begin{align*}
\text{if } u &\equiv_{PB} v \ast X + w, \\
X \notin \text{Var}(v, w), \\
v &\not\equiv_{PB} 0
\end{align*}
\]

\[
\begin{align*}
\frac{z \doteq \max(u)}{\{z \doteq \max(v^+ + w)\} \cup \{X \doteq v^>\}}
\end{align*}
\]

where \{X_1, \ldots, X_n\} are the only variables occurring in the terms \(v, w\) and \(v \neq 0\). We apply the variable elimination rule and obtain the new problem

\[
\{z \doteq \max(v^+ + w)\} \cup \{X_{n+1} \doteq v^- \ast X' + v^>\},
\]

with a new variable \(X'\).

Applying the induction hypothesis to \(z \doteq \max(v^+ + w)\) then yields in at most \(n\) steps the solved form

\[
\{z \doteq c, X_1 \doteq t_1, \ldots, X_n \doteq t_n\},
\]

with \(c \in \mathbb{Z}, t_i \in T(F_{psbool}, V_{bool})\), where \(c\) is the maximum value and \(\{X_1 \leftarrow v\} \circ \ldots \circ \{X_n \leftarrow t_n\}\) is a most general maximizer of \(v^+ + w\).

Using Lemma 19 and Lemma 20 we may conclude that \(c\) is also the maximum value of \(f\) and that \(\{X_1 \leftarrow v\} \circ \ldots \circ \{X_n \leftarrow t_n\} \circ \{X_{n+1} \doteq v^- \ast X' + v^>\}\) defines a most general maximizer of \(f\).

\[\square\]

5.2 The Special Version of the Basic Algorithm

Sometimes one is interested only in one special maximizer of a given pseudo-Boolean function and not in the most general maximizer. In this case, one can use a special version of the Basic Algorithm (see Figure 3).

For the special version of the Basic Algorithm an interesting complexity result was recently proved in [CHJ90]. In this paper it is shown that for some special class of pseudo-Boolean functions, namely the pseudo-Boolean functions of bounded tree-width, the special version of the Basic Algorithm runs in linear time.

**Definition 22** Let \(f(X_1, \ldots, X_n)\) be a pseudo-Boolean function in polynomial form. The co-occurrence graph \(G(f)\) has vertex-set \(V = \{X_1, \ldots, X_n\}\) and an edge between \(X_i\) and \(X_j, i \neq j\), if these variables occur simultaneously in at least one monomial of \(f\). The tree-width of \(f\) is the tree-width of the co-occurrence graph \(G(f)\).
The definition of the tree-width of a graph is rather technical and can be found in the appendix (see also [Arn85]).

**Theorem 23 ([CHJ90])** The Basic Algorithm has linear-time complexity when applied to pseudo-Boolean functions with bounded tree width.

More precisely: For every fixed $k \geq 0$, the Basic Algorithm can be implemented to run in $O(n)$ time on those pseudo-Boolean functions $f(x_1, \ldots, x_n)$, expressed in polynomial form, for which $(x_1, \ldots, x_n)$ is a $k$-scheme of $G(f)$.

If $g : \{0, 1\}^n \to \{0, 1\}$ is a Boolean function, then the equation $g(x_1, \ldots, x_n) = 1$ has a solution iff $\max(g(x_1, \ldots, x_n)) = 1$. This means that Boolean unification can be regarded as a special pseudo-Boolean optimization problem. In this case, the Basic Algorithm can be used to compute a special solution of the equation $g(x_1, \ldots, x_n) = 1$.

Moreover, for Boolean functions the Basic Algorithm specializes to a variant of Böttner-Simonis’ Boolean unification algorithm [BS87]. Therefore we get the following complexity result for Boolean unification.

**Corollary 24** For every fixed $k \geq 0$, Böttner/Simonis’ Boolean unification algorithm can be implemented to compute in $O(n)$ time a special solution of those Boolean unification problems $f(x_1, \ldots, x_n) = 1$, for which the Boolean function $f(x_1, \ldots, x_n)$ has tree-width at most $k$.

Because of the close relationship between the two algorithms, experiences that have been gained in the implementation of one of them have immediate consequences for the other one. For example, [CHJ90] compare two variable elimination orderings, which are related to the co-occurrence graph of the pseudo-Boolean function. The determination of a good elimination ordering is one of the main problems in the implementation of Böttner-Simonis’ algorithm. It has a great influence on the size of the most general unifiers that are computed. In this context, the orderings proposed in [CHJ90] may be very interesting. On the other hand, there exist rather sophisticated implementations of Böttner-Simonis’ algorithm. It is clear that the ideas which have been developed in this context should be used also in the pseudo-Boolean case.

### 6 Example

In this section we present a typical application of pseudo-Boolean constraints [HR68] that is simple enough to be considered here.

The problem is to assemble a radio set under the following conditions:

- Any one of the three types T1, T2, T3 of tubes may be utilized, but only one.
- The box may be either of wood W, or of plastic material P. When using P, dimensionality requirements impose the choice of T2, and as there is no place for a transformer F, a special power supply S is needed.
- T1 needs F.
- T2 and T3 need S (and not F).
The prices of the above components are

<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tubes T1</td>
<td>28 units</td>
</tr>
<tr>
<td>Tubes T2</td>
<td>30 units</td>
</tr>
<tr>
<td>Tubes T3</td>
<td>31 units</td>
</tr>
<tr>
<td>Transformer F</td>
<td>25 units</td>
</tr>
<tr>
<td>Special Power Supply S</td>
<td>23 units</td>
</tr>
<tr>
<td>Wood Box W</td>
<td>9 units</td>
</tr>
<tr>
<td>Plastic Material Box P</td>
<td>6 units</td>
</tr>
</tbody>
</table>

The other necessary components of the radio set cost

- 27 units, if the tubes T1 are utilized
- 28 units, if the tubes T2 are utilized
- 25 units, if the tubes T3 are utilized

The price of the make is 10 units for each set in all the cases and a set is sold at 110 units when it is enclosed in a plastic material box, and at 105 units in the other case. Which model is to be constructed in order to maximize the profit? The following program solves this problem.

\[
\text{assemble}(T1, T2, T3, F, S, W, P) :-
\]

\[
T1 + T2 + T3 = 1,
W + P = 1,
F + S = 1,
P \leq T2 \times S,  \quad \% P = 1 \implies T2 = S = 1
T1 \leq F,  \quad \% T1 = 1 \implies F = 1
T2 \leq S,  \quad \% T2 = 1 \implies S = 1
T3 \leq S.  \quad \% T3 = 1 \implies S = 1
\]

\[
\text{maximal-profit}(T1, T2, T3, F, S, W, M) :-
\]

\[
\text{max}( 110W + 105M - ( 28T1 + 30T2 + 31T3 + 25F + 23S \\
+ 9W + 6P + 27T1 + 28T2 + 25T3 + 10),
\]

\[
\text{assemble}(T1, T2, T3, F, S, W, M).
\]

Asking the query

\[- \text{maximal-profit}(T1, T2, T3, F, S, W, M).\]

yields the unique answer

\[T1 = 0, T2 = 0, T3 = 1, F = 0, S = 1, W = 1, M = 0.\]

7 Conclusion and Further Research

In this paper we have introduced a new constraint logic programming language \( CLP(PB) \) for logic programming with pseudo-Boolean constraints. The language is an instance of the constraint logic programming language scheme \( CLP(X) \) and has therefore the usual declarative and operational semantics of constraint logic programming languages. The main emphasis of the paper was on
the solution of pseudo-Boolean constraints. We showed that any pseudo-Boolean constraint has a
most general solution and gave an algorithm for pseudo-Boolean unification. Then we discussed the
problem of optimizing pseudo-Boolean functions. We presented the Basic Algorithm for pseudo-
Boolean optimization and discussed its relationship to the variable elimination method in Boolean
unification. Among others, we obtained a new complexity result for Boolean unification.

Several research directions can be outlined to continue this work. On the practical side, the
most important problem is to design an efficient constraint solver for pseudo-Boolean constraints.
This should be done using the techniques that have been developed for the Boolean case. The
next step is to extend this constraint solver by an algorithm for pseudo-Boolean optimization. On
the theoretical side, it would be interesting to find further classes of Boolean and pseudo-Boolean
functions that admit polynomial unification or optimization algorithms. Various such classes are
already known, however it is not clear which role they play in practical applications.

References

[Arn85] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded

[ASS+88] A. Aiba, K. Sakai, Y. Sato, D.J. Kawley, and R. Hasegawa. Constraint logic program-
ming language CAL. In Fifth Generation Computer Systems, Tokyo, 1988. Springer-

Workshop on Boolean Functions, Propositional Logic, and AI Systems. FAW Ulm,
Germany, 1989.


[CHJ90] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudo-boolean pro-

[Col87] A. Colmerauer. Introduction to PROLOG III. In 4th Annual ESPRIT Conference,

logic programming language CHIP. In Fifth Generation Computer Systems, Tokyo,


A Tree-Width of a Graph

Let $G$ be a simple and undirected graph. A vertex $v$ is simplicial in $G$ if the neighbors of $v$ induce a complete subgraph of $G$. For $k \geq 0$, a $k$-perfect elimination scheme of the graph $G$ is an ordering $(v_1, \ldots, v_n)$ of its vertices, such that $v_j$ is simplicial and has degree $k$ in the subgraph $G_j$ of $G$ induced by $\{v_j, v_{j+1}, \ldots, v_n\}$ for $j = 1, \ldots, n - k$. A graph is a $k$-tree if it has a $k$-perfect elimination ordering. A partial $k$-tree is any graph obtained by deleting edges from a $k$-tree. Any graph on $n$ vertices is a partial $n$-tree. The tree-width of a graph is the smallest value of $k$ for which the graph is a partial $k$-tree. For instance, trees and forests have tree-width at most 1, and series-parallel graphs have tree-width at most 2. Determining the tree-width of a graph is NP-hard. However, for every fixed $k$, there is a polynomial algorithm to decide if a graph has tree-width $k$. 

\[20\]