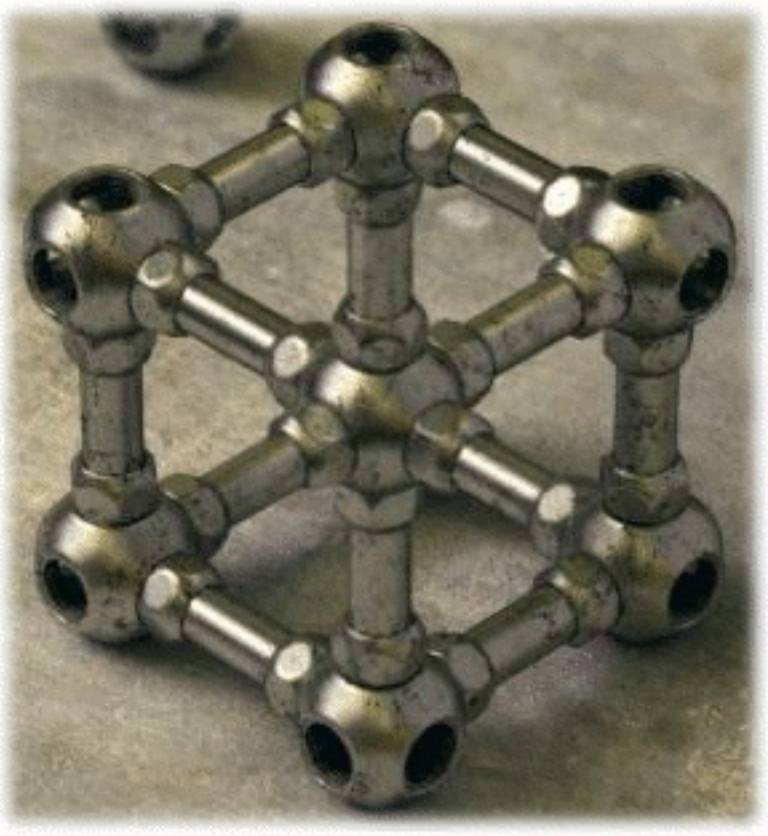

A Logic Based Algorithm for Solving Probabilistic Satisfiability [1]

Author(s):

Marcelo Finger

Glauber De Bona

[1] This work was supported by Fapesp Project LogProb, grant 2008/03995-5, São Paulo, Brazil.

A Logic Based Algorithm for Solving Probabilistic Satisfiability*

Marcelo Finger** and Glauber De Bona

Department of Computer Science (IME-USP)
Institute of Mathematics and Statistics
University of Sao Paulo — Brazil
{mfinger,debona}@ime.usp.br

Abstract. This paper presents a study of the relationship between probabilistic reasoning and deductive reasoning, in propositional format. We propose an algorithm to solve probabilistic satisfiability (PSAT) based on the study of its logical properties. Each iteration of the algorithm generates in polynomial time a classical (non-probabilistic) formula that is submitted to a SAT-oracle. This strategy is a Turing reduction of PSAT into SAT. We demonstrate the correctness and termination of the algorithm.

1 Introduction

Probabilistic satisfiability (PSAT) is a problem that allows for the joint application of deductive and probabilistic reasoning; this form of reasoning can be performed without any a priori assumptions of statistical independence. Such reasoning capabilities may be very useful in modeling complex systems, where the statistical interdependence between the components is not known.

PSAT original formulation is credited to George Boole and dates back to 1854 [2]. The problem has since been independently rediscovered several times (see [9, 10]) until it was introduced to the Computer Science and AI community by Nilsson [13] and was shown to be an NP-complete problem, even for cases where the corresponding classical satisfiability is known to be in PTIME [6]. Initial attempts to implement a solution led to the belief that the problem appeared to be very hard [14].

The PSAT problem is formulated in terms of a linear algebraic problem of exponential size. The vast majority, if not all, of algorithms for PSAT solving in the literature are based on linear programming techniques, such as column generation and matrix decomposition; see [10] for a good summary. There were also works covering extensions of PSAT with imprecise probabilities [11], and the discovery of some tractable fragments of PSAT [1], after which very little was published on algorithms for PSAT.

* This work was supported by Fapesp Thematic Project 2008/03995-5 (LOGPROB).
** Partially supported by CNPq grant PQ 304607/2007-0.

In the beginning of this century, very efficient solvers that dealt with the classical (non-probabilistic) satisfiability problem (SAT) became available [12, 7]. The SAT problem is a well studied problem, with known hard and easy instances and a phase-transition complexity distribution [5]. Efficient SAT-solvers enabled the solution of other NP-complete problems by translating them to a SAT instance and applying the solvers. Cook's theorem guarantees that there exists a polynomial-time translation from any NP-complete problem into SAT. However, no such translation from PSAT to SAT is found in the literature.

The aim of this paper is to study the logical properties of PSAT so as to present a *Turing reduction* of PSAT into SAT. A Turing reduction of PSAT into SAT is an algorithm to decide PSAT that employs a SAT-oracle [15].

The rest of the paper develops as follows. Section 2 presents the PSAT problem and proposes an *atomic normal form* to deal with it. The logical properties of PSAT are studied via a probabilistic entailment relation in Section 3, which allows for the presentation of a PSAT solving algorithm. Sections 4 and 5 demonstrate its correctness and that every step of the algorithm can be performed in polynomial time. Termination of the algorithm is shown in Section 6. Proofs have been omitted due to space restrictions.

2 The PSAT Problem

Probabilistic satisfiability consists in determining if the following configuration is consistent: a finite set of k classical propositional formulas $\alpha_1, \ldots, \alpha_k$ defined on n logical variables $\mathcal{P} = \{x_1, \ldots, x_n\}$, restricted by a set of probabilities imposed on the truth of these formulas, $p_1, \ldots, p_k$, such that $P(\alpha_i) = p_i, 1 \leq i \leq k$.

Formally, consider $\mathcal{L}$ the set of all propositional formulas, which may or may not be in clausal form. A propositional valuation v is initially defined over propositional variables, $v : \mathcal{P} \to \{0, 1\}$ and then is extended, as usual, to all formulas, $v : \mathcal{L} \to \{0, 1\}$. Consider the set V of 2^n propositional valuations $v_1, \ldots, v_{2^n}$ over the n variables. A *probability distribution over propositional valuations* $\pi : V \to [0, 1]$ is a function that maps every valuation to a value in the real interval $[0, 1]$ such that $\sum_{i=1}^{2^n} \pi(v_i) = 1$. The probability of a formula α according to distribution π is given by $P_\pi(\alpha) = \sum\{\pi(v_i) | v_i(\alpha) = 1\}$.

Define a $k \times 2^n$ matrix $A = [a_{ij}]$ such that $a_{ij} = v_j(\alpha_i)$. The *probabilistic satisfiability problem* is to decide if there exists a probability vector π of dimension 2^n such that obeys the *PSAT restrictions*:

$$\begin{aligned} A\pi &= p \\ \sum \pi_i &= 1 \\ \pi &\geq 0 \end{aligned} \tag{1}$$

The last two conditions force π to be a probability distribution. The condition $\sum \pi_i = 1$ is inserted as the first line of matrix A, that becomes a $(k + 1) \times 2^n$ matrix, and $p_{(k+1)\times 1}$ is now a vector with $p_0 = 1$. A *PSAT instance* is a set $\Sigma = \{P(\alpha_i) = p_i | 1 \leq i \leq k\}$, which is *satisfiable* iff its associated PSAT

restrictions (1) have a solution. If π is a solution to (1) we say that π satisfies Σ.

An important result of [6] guarantees that a solvable PSAT instance has a "small" witness.

Proposition 1 ([6]). *A PSAT instance $\Sigma = \{P(\alpha_i) = p_i | 1 \leq i \leq k\}$ has a solution iff there is a set of $k+1$ columns of A such that the resulting $(k+1) \times (k+1)$ system has a positive solution.*

The solution given by Proposition 1 serves as an NP-certificate for this instance, so PSAT is in NP. Furthermore, as *propositional satisfiability* (SAT) is a subproblem obtained when all $p_i = 1$, PSAT is NP-hard. So PSAT is NP-complete.

It follows from the Cook-Levin Theorem [3] that there must be a polynomial time reduction from PSAT to SAT. However, finding an efficient such reduction is neither obvious nor easy at all. But considering the fast advances in the construction of SAT solvers [12], it is a worthwhile task for two reasons. On the one hand, there is the basis for a general and relatively efficient implementation of PSAT which may render it useful in practical applications of moderate size, whose scale may increase with the advances in SAT solving. On the other hand, there is the understanding one gets from the relationship between probabilities and logic.

On the latter topic, one must add that the vast majority of PSAT solving initiatives, as surveyed in [10], concentrated in solutions based on variations of linear programming. But here we are interested in exploring the same problem via logical techniques. As PSAT poses no form of a priori probabilistic independence assumption, it is actually a form of probabilistic reasoning that is not very much explored in the literature.

First, some notation. If A is an $(k+1) \times (k+1)$ matrix, A^j represents its j-th column, and if $b_{(k+1)\times 1}$ is a column, $A[j := b]$ represents the matrix obtained by substituting b for A^j; $|A|$ is A's determinant.

2.1 The Atomic Normal Form

We say that a PSAT instance $\Sigma = \{P(\alpha_i) = p_i | 1 \leq i \leq l\}$ is in the *atomic normal form* if it can be partitioned in two sets, (Γ, Ψ), where $\Gamma = \{P(\alpha_i) = 1 | 1 \leq i \leq m\}$ and $\Psi = \{P(y_i) = p_i | y_i \text{ is an atom and } 1 \leq i \leq k\}$, with $0 < p_i < 1$, where $l = m + k$. The partition Γ is the SAT part of the normal form, usually represented only as a set of propositional formulas and Ψ is the *atomic probability assignment* part. An atomic PSAT instance is a PSAT instance in atomic normal form.

Theorem 1 (Atomic Normal Form). *Let $\Sigma = \{P(\alpha_i) = p_i | 1 \leq i \leq k\}$ be a PSAT instance. Then there exists an atomic PSAT instance (Γ, Ψ) such that Σ is a satisfiable iff (Γ, Ψ) is; the atomic instance can be built from Σ in polynomial time.*

The atomic normal form allows us to see a PSAT instance (Γ, Ψ) as an interaction between a probability problem (represented by Ψ) and a SAT instance Γ. Solutions to the instance can be seen as solutions to Ψ constrained by the SAT instance Γ. First, note that $\Psi = \{P(y_i) = p_i | 1 \leq i \leq k\}$ is a PSAT instance such that the columns of its corresponding matrix Y_Ψ are valuations over $y_1, \ldots, y_k$. However, for one solution to this instance to be a solution to the atomic instance (Γ, Ψ), this solution must be further constrained.

We say that a valuation v over $y_1, \ldots, y_k$ is consistent with a SAT instance Γ over variables $y_1, \ldots, y_k, x_1, \ldots, x_n$ if there is an extension of v over $y_1, \ldots, y_k, x_1, \ldots, x_n$ such that $v(S) = 1$ for every $S \in \Gamma$.

Lemma 1. *Let $\Psi = \{P(y_i) = p_i | 1 \leq i \leq k\}$. An atomic instance (Γ, Ψ) is satisfiable iff there is a $(k+1) \times k'$-matrix A_Ψ verifying the PSAT restrictions (1), $k' \leq k+1$, such that every column of A_Ψ is a valuation over $y_1, \ldots, y_k$ consistent with Γ.*

By Lemma 1, an atomic PSAT instance (Γ, Ψ) has a solution iff there is a matrix A satisfying conditions (2) below such that if $\pi_j > 0$ then $a_{1,j}, \ldots, a_{k,j}$ are Γ-consistent valuations for $y_1, \ldots, y_k$, $1 \leq j \leq k+1$:

$$\begin{bmatrix} 1 & \cdots & 1 \\ a_{1,1} & \cdots & a_{1,k+1} \\ \vdots & \ddots & \vdots \\ a_{k,1} & \cdots & a_{k,k+1} \end{bmatrix} \cdot \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_{k+1} \end{bmatrix} = \begin{bmatrix} 1 \\ p_1 \\ \vdots \\ p_k \end{bmatrix} \tag{2}$$

$\pi_j \geq 0$, A is non-singular,
$a_{i,j} \in \{0, 1\},\ 1 \leq i \leq k,\ 1 \leq j \leq k+1$

We will further assume, without loss of generality, that the input probabilities $p_1, \ldots, p_k$ are in decreasing order.

2.2 Relaxed PSAT

Given an atomic PSAT instance (Γ, Ψ) its associated *relaxed PSAT instance* is obtained by ignoring the restriction Γ, namely the instance $(\varnothing, \Psi)$.

Lemma 2. *Every relaxed PSAT instance is satisfiable.*

The matrix A_0 is called the *relaxed matrix of order* $k+1$ and consists of 1's on the diagonal and above it, and 0's below it. We assume we have a function *relaxedMatrixOfOrder*(k) that generates it.

The basic idea of the PSAT decision using a SAT-oracle is to start with a solution for the relaxed instance and try to substitute the columns of A_0 that are not consistent with Γ by others that are. The generation of these columns will be solutions to auxiliary SAT instances. If some auxiliary instance is unsatisfiable, so is the PSAT instance.

But before we can explain how the auxiliary SAT instances work, we need to introduce a few logical and algebraic tools.

3 A Probabilistic Entailment Relation

We propose a *probabilistic entailment relation*, $\mid\!\approx$, between an atomic probability assignment Ψ and a formula α, such that $\Psi \mid\!\approx \alpha$ iff the atomic PSAT instance $(\{\neg\alpha\}, \Psi)$ is (probabilistically) unsatisfiable. The intuition is that Ψ forces the $P(\alpha) > 0$; in fact, if $(\{\neg\alpha\}, \Psi)$ is unsatisfiable, this means that $P(\alpha) = 0$ is inconsistent with Ψ, but since $P(\alpha) \geq 0$ for any formula, it follows that $\Psi \mid\!\approx \alpha$ implies $P(\alpha) > 0$ for any probability distribution that satisfies Ψ.

Probabilistic entailment plays a special role in the study of probabilistic satisfiability, in the following sense.

Lemma 3. *Consider an atomic PSAT instance $\Sigma = (\Gamma, \Psi)$. Σ is satisfiable iff $\Gamma \cup \{\alpha\}$ is classically satisfiable for every α such that $\Psi \mid\!\approx \alpha$.*

Corollary 1. *$\Sigma = (\Gamma, \Psi)$ is (probabilistically) unsatisfiable iff there is a γ such that $\Psi \mid\!\approx \gamma$ and $\Gamma \models \neg\gamma$.*

Consider a PSAT solution of the form (2) and $\Gamma = \Gamma(y_1, \ldots, y_k; x_1, \ldots, x_n) = \Gamma_i(y; x)$. If we want to represent each column $A^j = \{a_{1,j}, \ldots, a_{k,j}\}$ as a set of variables $y^j = \{y^j_1, \ldots, y^j_k\}$, we have that $\Gamma(y^j; x^j)$, $1 \leq j \leq k+1$, have all disjoint sets of variables and have all to be jointly satisfiable. If we have a set of classical conditions $C = \{\alpha_1, \ldots, \alpha_c\}$ such that $\Psi \mid\!\approx \alpha_i$, $1 \leq i \leq c$, no matter how big the number of conditions c is, all conditions of C have to be satisfied by one of at most $k+1$ valuations, which are columns of the matrix A. Therefore, $\alpha_i(y^1; x^1) \vee \ldots \alpha_i(y^{k+1}; x^{k+1})$ must hold for each $\alpha_i \in C$. Furthermore, all these conditions have to be jointly satisfiable, as there are only $k+1$ valuations allowed in a solution of the form (2). We have thus proved the following.

Lemma 4. *Let C be a set of Ψ-entailed formulas. Then the following set of formulas must be jointly satisfiable:*

$$\begin{array}{l} \{\Gamma(y^j; x^j) | 1 \leq j \leq k+1\} \cup \\ \{\alpha(y^1; x^1) \vee \ldots \vee \alpha(y^{k+1}; x^{k+1}) | \alpha \in C\} \end{array} \tag{3}$$

We assume there is a function *satFormulaFromConditionSet*(C) that given a set of Ψ-entailed formulas, generates a formula φ of the format (3) to be sent to a SAT oracle. If φ is unsatisfiable, by Corollary 1 the initial PSAT instance is also unsatisfiable. Otherwise it outputs a valuation v from which we can assemble a $(k+1) \times (k+1)$ matrix B extracting the value v attributes to y^j, $1 \leq j \leq k+1$; the first line of B is all 1's. We assume there is a function *assembleMatrix*(v) that constructs such matrix.

Note that B may not satisfy conditions (2) as, for instance, it may be singular in the case that all conditions of C are satisfied by a $k' < k+1$ columns, so that $v(y^{k''}) = v(y^{k'})$ for $k'' > k'$. So we have to merge the columns of B with those of A to generate a new matrix satisfying conditions (2). We are now in a position to present an outline of the PSAT-solver algorithm.

3.1 Algorithm

Based on Lemma 4 we are in a position to present the outline of the algorithm that decides a PSAT instance employing a number of applications of the SAT problem, which is presented in Algorithm 3.1.

Algorithm 3.1 PSAT via SAT

Input: An atomic normal form PSAT instance $\Sigma = (\Gamma, \Psi = \{P(y_i) = p_i | 1 \leq i \leq k\})$.
Output: No, if Σ is unsatisfiable. Or a solution matrix A satisfying (2).

```
 1: i := 0
 2: A_0 := relaxedMatrixOfOrder(k + 1)
 3: C_0 := ∅ //No initial conditions
 4: while A_i is not a solution for (Γ, Ψ) do
 5:    C_{i+1} = C_i ∪ conditionsFromMatrix(A_i)
 6:    φ := satFormulaFromConditionSet(C_{i+1})
 7:    if φ is SAT-solvable with valuation v then
 8:       B := assembleMatrix(v)
 9:       A_{i+1} = merge(A_i, B) //Invariant: A_{i+1} satisfies (2)
10:       increment i
11:    else
12:       return No
13:    end if
14: end while
15: return A_i
```

So, the basic idea is to start every iteration with a matrix A_i satisfying (2), use it to generate a set of Ψ-entailed formulas, generate a formula φ from it and submit it to a SAT-oracle; if φ is unsatisfiable, so is the initial instance Σ. Otherwise, obtain a matrix B satisfying all conditions of φ and merge it with A_i, generating A_{i+1} satisfying (2) and check if its a solution. If it is, return A_{i+1}; otherwise, repeat this process.

To check for a solution (line 4), we inspect the vector π in the invariant (2). A solution is found when for all components $\pi_j > 0$, column A_i^j represents a Γ-consistent valuation. If $\pi_j = 0$, we need not enforce A_i^j to be a Γ-consistent valuation, as any other column can be substituted for it; this represents the case where a solution for (1) can be found with rank less than $k + 1$.

To complete the description of Algorithm 3.1, we still have to clarify the following points:

- Line 5: how to obtain a set of Ψ-entailed formulas from a matrix satisfying (2).
- Line 9: how to merge a matrix A satisfying (2) with another, generating a third one that still satisfies (2), keeping the invariant a line 9.
- Line 4: how the loop halts in a finite number of steps.

4 Linear Algebra and Logic

In this section we examine some relations between Linear Algebra and Logic. In particular, we obtain Ψ-entailed formulas from a matrix satisfying (2).

4.1 Linear restrictions as formulas

A linear restriction over variables $x_1, \ldots, x_k$ is a condition of the form

$$a_1 \cdot x_1 + \cdots a_k \cdot x_k \quad op \quad c \tag{4}$$

where $a_1, \ldots, a_k, c \in \mathbb{Q}$ and $op \in \{<, \leq, >, \geq, =, \neq\}$. If we further restrict the variables x_i such that $x_i \in \{0, 1\}$, $1 \leq i \leq k$, a linear restriction LR of the form (4) can be seen as a formula, in the sense that a valuation v such that $v(x_i) \in \{0, 1\}$ *satisfies* LR if the restriction is verified when we substitute $v(x_i)$ for x_i in (4). This view of linear-restrictions-as-formulas was developed in the area of linear programming [8], and has been used in pseudo-boolean inference methods [16, 4].

Proposition 2. *Every linear restriction LR can be represented by a set of clauses Δ_{LR} such that LR is satisfiable iff Δ_{LR} is; the number of atoms and clauses in Δ is bound by polynomial in k.*

For example, the linear restriction

$$2x_1 - x_2 + x_3 \geq 2$$

when restricted to $x_1, x_2, x_3 \in \{0, 1\}$ can be seen as representing the clausal form formula $x_1 \wedge (\neg x_2 \vee x_3)$.

Proposition 2 allows us to deal with linear restrictions as a formula, without worrying about an exponential explosion, as they can be brought to clausal normal form in polynomial time.

4.2 Logical restrictions for column substitution

We now study the logical conditions for substituting columns of a given matrix that satisfy the invariant condition (2). Given a matrix A satisfying (2), we will construct a matrix $C_{k+1 \times k+1} = [c_{i,j}]$ of logical conditions.

Recall that A^j represents A's j-th column, and if $b_{(k+1)\times 1}$ is a column, $A[j := b]$ represents the matrix obtained by substituting b for A^j.

Let $y = [1\ y_1\ \ldots\ y_k]'$ be a column of variables. If A satisfies (2), and we want to substitute its j-th column by $b = [1\ b_1\ \ldots\ b_k]'$, to keep (2) invariant, $A[j := b] \cdot \pi = p$, then necessarily, $|A[j := b]| \neq 0$. This condition may be stated syntactically by the linear-restriction-as-formula $|A[j := y]| \neq 0$, where the logical restrictions on y represent the algebraic restrictions on b; in fact, this restriction is linear for *only one column* is being substituted by variables. This

may already be expressed as a logical formula. However, more expressivity can be obtained from *Cramer's Rule*, according to which:

$$\pi_j = \frac{|A[j := y][j := p]|}{|A[j := y]|} \geq 0 \tag{5}$$

Note that $A[j := y][j := p] = A[j := p]$ contains no variables. Let $\text{sign}(x) = -1$, if $x < 0$, and $+1$ otherwise. If $|A[j := p]| \neq 0$, then $\pi_j > 0$, so (5) yields

$$sign(|A[j := p]|) \cdot |A[j := y]| > 0 \tag{6}$$

which is a linear restriction. In the matrix of logical conditions C we make the diagonal elements represent these conditions. Let $A' = A[j := p]$, then

$$c_{j,j} = \begin{cases} sign(|A'|) \cdot |A[j := y]| > 0, & \text{if } |A'| \neq 0 \\ |A[j := y]| \neq 0, & \text{if } |A'| = 0 \end{cases} \tag{7}$$

Note that the $c_{j,j}$ enforce that the column that substitutes A^j, that is non-singular, will make the resulting matrix also non-singular. The other elements of matrix C are also obtained using Cramer's rule. Fix column j and consider a column $i \neq j$; we have

$$\pi_i = \frac{|A[j := y][i := p]|}{|A[j := y]|} \geq 0 \tag{8}$$

As $c_{j,j}$ enforces $|A[j := y]| \neq 0$. If we further have that $|A[j := p]| \neq 0$, (6) and (8) yield that

$$sign(|A[j := p]|) \cdot |A[j := y][i := p]| \geq 0,$$

which is a linear restriction and can be expressed by a formula $\chi_{i,j}$. When $|A[j := p]| = 0$ we need to consider two cases, namely $A[j := y] > 0$ and $A[j := y] < 0$; as both are linear restrictions that can be expressed as formulas, let us abbreviate those formulas as ψ_j and ξ_j, respectively. Now consider the following linear conditions, $|A[j := y][i := p]| \geq 0$ and $|A[j := y][i := p]| \leq 0$ and let us abbreviate the formulas that encode those conditions as $\psi_{i,j}$ and $\xi_{i,j}$; it is clear that when ψ_j holds, we want $\psi_{i,j}$ to hold, and when ξ_j holds we want $\xi_{i,j}$ to hold. In the matrix of logical conditions C we make the non-diagonal elements represent these conditions. Let $A' = A[j := p]$ and $i \neq j$, then

$$c_{i,j} = \begin{cases} \chi_{i,j}, & \text{if } |A'| \neq 0 \\ (\psi_j \wedge \psi_{i,j}) \vee (\xi_j \wedge \xi_{i,j}), & \text{if } |A'| = 0 \end{cases} \tag{9}$$

The elements of the condition matrix $C = [c_{i,j}]$, $1 \leq i, j \leq k + 1$, are given by (7) and (9) and are all logical formulas. Each column C^j is constructed concentrating on its corresponding column A^j and the restrictions on π_i, $1 \leq i \leq k + 1$. The following is a central property of the elements of matrix C.

Lemma 5. *Consider a matrix of conditions $C = [c_{i,j}]$ constructed as above based on a matrix A satisfying (2), $A \cdot \pi = p$. If $\pi_j > 0$ then $\Psi \models c_{i,j}$, $1 \leq i, j \leq k+1$.*

Lemma 5 states that the elements of the condition matrix C are conditions of the form needed in line 5 of PSAT-solving Algorithm 3.1. So we define

$$\mathit{conditionsFromMatrix}(A) = \{c_{i,j} \in C\} \cup \mathit{haltingCondition}(A) \ . \qquad (10)$$

where $\mathit{haltingCondition}(A)$ will be developed in Section 6.

5 Merging

Merging consists taking a matrix A satisfying the invariant conditions (2) and a matrix B whose columns represent Γ-consistent valuations and generating a new matrix A' that incorporate "some" of B columns such that A' satisfies the invariant conditions (2). The following lemma tells us how to merge a single column.

Lemma 6. *Let A be a matrix satisfying the invariant condition (2) and let $b = [1\ b_1\ \cdots\ b_k]'$ be a column such that $b_i\{0,1\}, 1 \leq i \leq k$. Then there always exists a column j such that $A[j := b]$ satisfies the invariant conditions (2).*

The proof of Lemma 6 gives a procedure to merge a single column b with a given matrix A preserving invariant condition (2). Consider matrix $B = [b_1 \ldots b_{k+1}]$ as a sequence of columns. We define the $\mathit{merge}(A, B)$, used in line 9 in Algorithm 3.1, as the successive merging of each column b_i. As each merging step, by Lemma 6, preserves the invariant condition (2), so does the output of $\mathit{merge}(A, B)$.

As noted above, the substituted column always exists in the merge process, but to ensure termination we assume that the merge procedure always chooses to substitute a column that is not Γ-consistent, if one is available.

6 Termination

To guarantee termination the halting formula will force at least one Γ-inconsistent formula to be substituted at each iteration of Algorithm 3.1.

Let A be a $\{0,1\}$-matrix satisfying invariant conditions (2), and let $C = [c_{i,j}]$ be its corresponding condition matrix as in Section 4.2. Then any column y substituting A^j has to satisfy all the conditions in column C^j; let $d_{i,j}(y)$ be defined as

$$d_{i,j}(y) = c_{i,j}(y) \wedge c_{j,j}(y) \ .$$

As $c_{j,j}$ makes $|A[j := y]| \neq 0$, $d_{i,j}(y)$ guarantees that $A[j := y]$ is non-singular. Considering the merging method of Section 5, if after ℓ substitutions every $d_{i,j}$, $1 \leq i \leq k+1$, was satisfying by a substituting column, then column j has been

substituted in the merging process. The halting formula guarantees that j is a Γ-inconsistent column of A.

Suppose there are $\ell - 1$ Γ-consistent columns in A, and consider the set $S = \{j | A^j$ is a Γ-inconsistent columns with $\pi_j > 0\}$. Then the halting condition is

$$haltingCondition(A) = \bigvee_{j \in S} \bigwedge_{i=1}^{k+1} d_{i,j}(y^1) \vee \ldots \vee d_{i,j}(y^\ell) \ . \tag{11}$$

Theorem 2. *Algorithm 3.1 terminates.*

7 Conclusions and future work

We have presented an algorithm that decides PSAT using a SAT-oracle, a theoretical work with potential practical applications.

Future work contemplates an implementation of a PSAT solver based on the algorithm presented here, which will employ an existing SAT-solver (such as zchaff [12]) as an implementation of the SAT oracle. We plan to make this PSAT-solver available as open-source software.

References

1. Kim Andersen and Daniele Pretolani. Easy cases of probabilistic satisfiability. *Annals of Mathematics in Artificial Intelligence*, 33(1):69–91, 2001.
2. George Boole. *An Investigation on the Laws of Thought.* Macmillan, London, 1854.
3. S. A. Cook. The complexity of theorem-proving procedures. In *Third Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158. ACM, 1971.
4. Heidi E. Dixon and Matthew L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *AAAI/IAAI*, pages 635–640, 2002.
5. I. P. Gent and T. Walsh. The SAT phase transition. In *ECAI94 – Proceedings of the Eleventh European Conference on Artificial Intelligence*, pages 105–109, 1994.
6. George Georgakopoulos, Dimitris Kavvadias, and Christos H. Papadimitriou. Probabilistic satisfiability. *Journal of Complexity*, 4(1):1–11, 1988.
7. E. Goldberg and Y. Novikov. Berkmin: A Fast and Robust SAT Solver. In *Design Automation and Test in Europe (DATE2002)*, pages 142–149, 2002.
8. R. Gomory. An algorithm for integer solutions to linear programs. In *Recent Advances in Mathematical Programming*, pages 69–302. McGraw-Hill, 1963.
9. T. Hailperin. Prob. logic. *N. D. Journal of Formal Logic*, 25(3):198–212, 1984.
10. P. Hansen and B. Jaumard. Probabilistic satisability. Technical Report, Les Cahiers du GERAD G-96-31, École Polytechique de Montréal, 1996.
11. P. Hansen *et al.* Probabilistic satisfiability with imprecise probabilities. In *1st Int. Symposium on Imprecise Probabilities and Their Applications*, pp. 165–174, 1999.
12. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.
13. Nils Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.
14. N. Nilsson. Probabilistic logic revisited. *Artificial Intelligence*, 59(1–2):39–42, 1993.
15. Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.
16. Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Inf. Process. Lett.*, 68(2):63–69, 1998.