

Approximated Probabilistic Answer Set Programming

Eduardo Menezes de Moraes

Department of Computer Science
University of São Paulo
São Paulo, Brazil

Abstract. This paper is a work in progress showing our search for a modification of Probabilistic Answer Set Programming (PASP), a technique that allows modeling complex theories and checking its satisfiability with respect to a set of probabilistic data, to obtain approximated solutions.

Keywords: logic programming; probabilistic logic; probabilistic satisfiability (PSAT); answer set programming (ASP).

1 Introduction

This paper is a work in progress. Probabilistic Answer Set Programming (PASP) is an extension of Answer Set Programming (ASP, [7,8]) to deal with probabilistic constraints. It was introduced by the authors in [9,10]. In this paper we hope to extend PASP further by developing an algorithm to search for approximated answers.

PASP is based on the interaction between logic and probability originally due to Boole [4] and which is now known as Probabilistic Logic [12] or Probabilistic Satisfiability (PSAT, [6]). Recent work has shown that PSAT has industrial applications [5], but for its use in larger problems a better modeling tool is necessary. One possible such tool is PASP. A toy example of the type of problem that can be tackled with PSAT is shown in Example 1.

Example 1. Consider Figure 1 and let's assume that we want to go from vertex 1 to vertex 6. However we want to limit the number of times we pass through edge (1,3) and (3,4) to 50% and 40% respectively. Is there a probability distribution over the paths in this graph such that the marked edges are only utilized the appropriated number of times? What if the probabilities are modified to 20% and 40% respectively?

This problem can be formulated as a Probabilistic Satisfiability Problem, however its formulation is tedious and results in a large formula. On the other hand the problem of finding paths in a graph is much more simple to formulate as an ASP program.

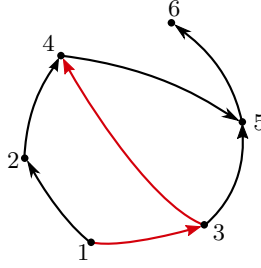


Fig. 1. A directed graph with several paths between 1 and 6

ASP is a variation of Logic Programming that employs stable model semantics to deal with negation-as-failure, and has been successfully employed in applications ranging from typical AI problems to Software Engineering and Computational Biology.

PASP should be a tool for finding solutions to problems that combines “hard” constraints, that is, logical constraints that can never be broken in a solution, and “soft” constraints that could be broken but ideally would be satisfied. In [5] we see a similar application using PSAT. Since it is easier to model complex theories in ASP than in SAT, with PASP would allow modeling larger and more difficult problems.

Since PASP was introduced to the literature recently, we present here it’s main points. Part of the text of this article is taken from [10], where the authors first presented this framework that we now seek enhance.

The rest of this work is developed as follows. Background on ASP and PSAT is presented briefly in Sections 2 and 3. More details can be found in [9]. The PASP problem is presented in Section 4 and a it’s modifications for approximated answers is presented in Section 5.

2 Answer Set Programming

Answer Set Programming is a non-monotonic, declarative programming paradigm for solving hard combinatorial problems. Introduced in [8] as a proper logic programming paradigm, **Answer Set Programming** (ASP) derives from the concept of **Answer Sets**, a concept created by Gelfond and Lifschitz in [7] in an attempt to clarify the semantics of negation in logic programming languages.

ASP program can have variables and functions, but it must be grounded before it’s *Answer Sets* can be found. A Grounded ASP program is a finite set of rules of the form $h \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$, where h and each L_i is a literal (possibly with classical negation, also called explicit negation, $\neg a$). The symbol *not* represents **default negation** or **negation as a failure to prove**. In a rule r , “ h ” is the **head**, represented by $Head(r)$ and “ $L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$ ” is the **body**, represented by $Body(r)$. The literals $L_1, \dots, L_m$ form the **positive body**, $Body^+(r)$ and $L_{m+1}, \dots, L_n$ form

the **negative body** of the rule, $Body^-(r)$. Rules without heads are called **restrictions**.

A set of literals M is a **model** of program P if for every rule r of the program, if $Body^+(r) \subseteq M$ and $Body^-(r) \not\subseteq M$, then $Head(r) \in M$. A model can't have both an atom a and $\neg a$. We say a model M^* is a **minimal model** if it's minimal relative to set inclusion, that is, there is no model M such that $M \subset M^*$. Programs where in all rules the negative body is empty have an unique minimal model [14]. That is not true in general for an arbitrary program.

The definition of *Answer Set* of a grounded ASP program uses the concept of the **reduction of a program**.

Definition 1 (from [7]). Let M be a finite set of atoms of P , the program P_M , obtained from P by removing:

- all the rules that have a literal A in their negative body if $A \in M$;
- the negative body of the remaining rules

is called **reduction of P by M** .

We can verify that the reduction of a program P results in a program where in all rules the negative body is empty. Therefore, there is always an unique minimal model for the reduction of a program.

Definition 2. Let P_M be the reduction of the program P by M , M is an **Answer Set** of P if the minimal model of P_M is M .

An useful extension to the traditional ASP framework is the **weight constraint rule**, introduced in [11]. A **weight constraint** has the form:

$$L \leq \{h_1 = w_1, \dots, \text{not } h_n = w_n\} \leq U. \quad (1)$$

where h_i are literals or default negated literals, L and U are integers, called **lower limit** and **upper limit** respectively, and $w_1, \dots, w_n$ are called **weights** associated to a literal, also represented by $weight(h_i)$.

Intuitively, a weight constraint is satisfied if the sum of the weights of the satisfied literals is between L and U .

Example 2. Consider the weight constraints

$$C_1 : \quad 2 \leq \{a = 1, b = 2, \text{not } c = 1\} \leq 3$$

$$C_2 : \quad 1 \leq \{\text{not } a = 2, b = 1, c = 1\} \leq 2$$

The set $\{a, b\}$ satisfies C_2 and not C_1 .

A **weight rule** has the form:

$$C_0 \leftarrow C_1, \dots, C_n \quad (2)$$

where $C_0, \dots, C_n$ are weight constraints and in C_0 there are no default negated literals. A weight constraint rule is satisfied if when $C_1, \dots, C_n$ are satisfied, C_0 also is. The reduction (Definition 1) of a weight constraint is defined accordingly.

A result that should be observed is:

Theorem 1 (from [9]). *Given an ASP program S with Ψ as the set of all Answer Sets, by adding a new constraint rule, the new program S' will have Answer Sets $\Psi' \subseteq \Psi$.* $\square$

This theorem shows that by adding constraints to an ASP program we do not add new *Answer Sets*, we only remove those that violate the constraint. This holds for both regular constraint rules and weight rules with an empty head.

3 PSAT

The Probabilistic Satisfiability problem (PSAT), first shown in 1854 by George Boole [4] and later rediscovered by several researchers, is the problem of deciding if a set probabilities associated to set of logic formulas is satisfiable, as defined below.

Let's consider the classic propositional logic in the usual way. A PSAT instance is given by a set of formulas in propositional logic over a set of n variables, $S = \{s_1, \dots, s_k\}$, and a set of probabilities $P = \{p_1, \dots, p_k\}$ with $0 \leq p_i \leq 1$ for all i .

A **discrete probability distribution** over an enumerable set X is a function π that associates each element of this set with a probability in such a way that $\sum_{x \in X} \pi(x) = 1$.

Given a probability distribution over all S set's formulas' interpretations, we can define the **probability of a formula** as the sum of the probabilities of the interpretations that satisfy this formula. Formally, if π is a probability distribution over the set of interpretations, I_S , and $i(s) = 1$ iff i is a valid interpretation of the formula $s \in S$, formula s 's probability is:

$$p(s) = \sum_{i \in I_s} \{\pi(i) | i(s) = 1\} \quad (3)$$

Intuitively, a formula's probability is sum of the probabilities of every "possible world" where the formula holds.

Definition 3. *Let $I_S = \{i_1, \dots, i_{2^n}\}$ be the set of possible interpretations over the variables of S , we say that the probability association $p(s_i) = p_i$ is **satisfiable** if there exists a probability distribution π over I_S such that Equation (3) holds.*

Example 3. Consider the following PSAT instance:

$$\begin{aligned} P(a \vee b \vee c) &= 1 \\ P(a \wedge b) &= 0.61; P(a \wedge c) = 0.60; P(b \wedge c) = 0.59 \end{aligned}$$

And consider the interpretations $v_1 = \{a = b = c = 1\}$, $v_2 = \{a = b = 1; c = 0\}$, $v_3 = \{a = c = 1; b = 0\}$ and $v_4 = \{a = 0; b = c = 1\}$.

The probability distribution π that associates the following values to these interpretations: $\pi(v_1) = 0.4; \pi(v_2) = 0.21; \pi(v_3) = 0.2; \pi(v_4) = 0.19$ is a solution

to this PSAT instance, as we can verify by replacing these values in the equations below, that are instances of the Equation 3 for each formula:

$$\pi(v_1) + \pi(v_2) + \pi(v_3) + \pi(v_4) = 1 \quad (4)$$

$$\pi(v_1) + \pi(v_2) = 0.61 \quad (5)$$

$$\pi(v_1) + \pi(v_3) = 0.6 \quad (6)$$

$$\pi(v_1) + \pi(v_4) = 0.59 \quad (7)$$

This system of equations can be seen in matricial form:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.4 \\ 0.21 \\ 0.2 \\ 0.19 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.61 \\ 0.60 \\ 0.59 \end{bmatrix} \quad (8)$$

A PSAT instance can be expressed as a linear programming problem defining the matrix $A_{k \times 2^n} = [a_{ij}]$, such that $a_{ij} = 1$ iff an interpretation j satisfies the formula i and also defining the vector $p_{k \times 1} = [p_i]$. A PSAT instance is satisfied if there exists a vector π that is a solution to:

$$\begin{aligned} A\pi &= p \\ \pi &\geq 0 \\ \sum \pi_i &= 1 \end{aligned} \quad (9)$$

The last restriction can be omitted by prefixing in A and p an entire line of ones.

If this system has a solution, thanks to Carathodory's lemma [13], it has a solution with only $k + 1$ nonzero elements, making PSAT a *NP*-complete problem.

4 PASP

Probabilistic Answer Set Programming (PASP) [9, 10] is an extension of Answer Set Programming with information about probability. In Section 5, we further develop this framework with approximated probabilities.

4.1 Definition

Like PSAT, PASP is a decision problem where we must decide whether a set of probabilities is **satisfiable** by the rules of a program. The input of a PASP problem is a grounded ASP program, S , without classical negation and with a Herbrand base $HB_S = \{a_1, \dots, a_n\}$, and a set of probabilities $P = \{p_1, \dots, p_k\}$, $k \leq n$, associated to a subset of HB_S . We assume the probabilities are rational numbers.

The formal definition of satisfiability in PASP resembles PSAT's concept of satisfiability.

Definition 4. Let $2^{HB_S} = \{v_1, \dots, v_N\}$ be the set of all subsets of HB_S (the candidates to Answer Set of S).

We say a set of probabilities $P = \{P(a_i) = p_i | 1 \leq i \leq k\}$ is **satisfied** by S if there is a probability distribution π over 2^{HB_S} where

$$p_i = \sum \{\pi(v_l) | a_i \in v_l \text{ and } v_l \text{ is Answer Set of } S\}$$

Note that, unlike PSAT, the probabilities in PASP refer only to literals, not to arbitrary formulas. However, it is possible to associate probabilities with arbitrary rules by introducing new literals in the head of these rules. This can be seen in more details in [9].

Example 4. Consider the following ASP program:

$$\begin{aligned} 1 \leq \{useEdge(X, Y) = 1 \text{ for each } edge(X, Y)\} \leq 1 &\leftarrow vertex(X), pathTo(X). \\ &pathTo(1). \\ pathTo(Y) &\leftarrow pathTo(X), useEdge(X, Y), edge(X, Y). \\ &\leftarrow not\ useEdge(6). \end{aligned}$$

Taken together with a description of the graph in Figure 1, this ASP program finds all the paths from vertex 1 to vertex 6. The *Answer Sets* of this program are $I_1 = \{useEdge(1, 2), useEdge(2, 4), useEdge(4, 5), useEdge(5, 6)\}$, $I_2 = \{useEdge(1, 3), useEdge(3, 5), useEdge(5, 6)\}$, and $I_3 = \{useEdge(1, 3), useEdge(3, 4), useEdge(4, 5), useEdge(5, 6)\}$.

If we use the set of probabilities $p(useEdge(1, 3)) = 0.5$ and $p(useEdge(3, 4)) = 0.4$ we have a PASP instance.

This particular instance is consistent. Consider π such that $\pi(I_1) = 0.5$, $\pi(I_2) = 0.1$, $\pi(I_3) = 0.4$, and $\pi(v_i) = 0$ for any other v_i , we can verify that π meets the satisfiability criteria.

We can write a PASP problem in binary matrix form in such a way that each column of the matrix represent a subset of HB_S , i.e., a candidate to *Answer Set* of S . The value of each line indicates the presence or absence of a atom in the subset.

This matrix is defined as $A_{k \times 2^{\|HB_S\|}} = [a_{ij}]$, such that $a_{ij} = 1$ if the j -th atom subset contains the i -th atom; we force $\pi_j = 0$ if the j -th column of A is not an *Answer Set* of P . This way the criterion for deciding the satisfiability of a PASP instance becomes:

$$\begin{aligned} A\pi &= p \\ \pi &\geq 0 \\ \sum \pi_i &= 1 \end{aligned} \tag{10}$$

Once again we can eliminate the last criteria by adding a line of ones.

Theorem 2. *If there exists an solution for a PASP instance, there is a solution with at most $k + 1$ nonzero elements in π .*

This Proposition follows directly from Carathodory's Lemma.

4.2 Related work

There have been other works that extend Answer Set Programming with probabilities, such as **P-log** [1, 2].

The way P-log tackles probabilities is radically different from PASP's. P-log works with causal probabilities (intuitively, probabilities that are independent of any factor other than its cause), observations, and actions. Its focus is knowledge representation and inference mechanism over Bayesian networks.

On the other hand, PASP's focus is hard and soft constraints and there is no probability independence or dependence hypothesis.

Therefore, even though both PASP and P-log are both methods of extending ASP with probabilities, their focus are very different and they should be used for different tasks.

4.3 Resolution methods

Since a PASP instance can be written as a Linear Programming problem without a cost function, we can reduce the problem of solving a PASP instance to the problem of finding a basic viable solution where the columns correspond to *Answer Sets*. This can be done with the first phase of the Simplex algorithm [3].

We introduce $n+1$ artificial variables, ψ , where n is the size of the probability vector. Note that those are not variables from ASP's point of view, but from the Linear Programming point of view.

In the Simplex algorithm we use the Identity matrix as a basic matrix. While it's possible to use it, a better choice in dealing with probability constraints is a matrix of form (11).

$$I^* = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 0 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \quad (11)$$

Since the matrix $[I^*|\pi]$ is in Echelon form [16], the system always has a solution. If the probability vector is decreasingly ordered, this solution is non-negative.

We need a way to select columns to enter the base so that we can minimize the probability mass associated to the artificial variables that are not *Answer Sets*, $\sum \psi_i$. In the traditional simplex algorithm, all the columns are enumerated, but in our case, since possibly there is an exponential number of columns, we just keep track of the base. So we must generate an *Answer Set* with negative reduced cost.

To say an *Answer Set* j has a negative reduced cost is equivalent to say that $\bar{c}_j = c_j - c'_B B^{-1} A_j < 0$. Since $c_j = 0$ for all *Answer Sets* we only need that

$$c'_B B^{-1} A_j > 0 \quad (12)$$

To obtain such *Answer Set* we generate a new ASP program S' in whose *Answer Sets* also are *Answer Sets* of S and furthermore, the reduced cost is negative. Because of Theorem (1), if we can express this requirement as a set of restrictions, we can simply concatenate S with these restrictions.

These “reduced cost” restrictions can be expressed with weight constraints or by expressing the Inequality (12) as SAT formulas, as can be seen in [15]. Details of both methods are available in [9].

Assuming the existence of a function *SelectColumn* that uses one the described methods to obtain a column with reduced cost, the Algorithm 1 presents a resolution method.

Algorithm 1 Method for solving PASP

Require: An ASP Program S , a set of k positive literals l , and a set of k probabilities,

```

 $p.$ 
 $B \leftarrow I_{(k+1) \times (k+1)}^*$  /* Initial Base */
 $c_B \leftarrow [1 \dots 1]_{k+1}$ 
for  $j = 1$  to  $k$  do
    if Satisfied( $B_j, S$ ) then
         $c_{B_j} \leftarrow 0$ 
    end if
end for
 $\pi_B \leftarrow B^{-1}p$  /* Initial Basic Feasible Solution */
while  $c'_B \pi_B > 0$  do
     $z \leftarrow \text{SelectColumn}(S, l, B, c_B)$ 
    if  $z = \emptyset$  then
        return UNSAT
    end if
    ChangeBase( $B, z, \pi, c_B$ )
     $\pi \leftarrow B^{-1}p$  /* New Basic Feasible Solution */
end while
return  $B, \pi_B$ 

```

The function *ChangeBase* replaces some column of B with column z and updates the cost vector, c_B . This is a standard procedure in linear programming algorithms and can be interpreted as the *Answer Set* represented by column z being assigned a non-zero probability.

Satisfied verifies if the column B_j represents an *Answer Set*. This is made easier because the values in each column represent the values of the variables in the corresponding set. But since not all literals are associated with probabilities, to implement *Satisfied* we cannot simply verify if a set is an *Answer Set* of S . Instead we must use an ASP Solver to find an *Answer Set* with or without a certain literal. Details can be seen in [9, 10].

The main loop of Algorithm 1 persists while $c'_B \pi$ is positive. It stops if the problem is unsatisfiable or when a zero cost has been reached. An implementation

of this algorithm utilizing weight rules is available at the repository `git://gitorious.org/pasp/pasp-asp.git` under the GPLv3 license.

5 Approximations

PASP in it's current form can be used in several applications, such as those presented in [5], where we have both hard constraints and probabilistic constraints.

However, some times we may encounter PASP instances that, even though are unsatisfiable, we would be interested in minimizing the error, that is, finding a probability distribution over the instances' *Answer Sets* such as the probability of each literal is the closest possible to the given probabilities.

Definition 5. *Given a probability distribution π , we define the **error of π** as*

$$E(\pi) = \sum_{i \in P} \left| p_i - \sum \{ \pi(v_l) | a_i \in v_l \text{ and } v_l \text{ is Answer Set of } S \} \right|$$

Consider the following example:

Example 5. Let us consider the same program from Example 4 but now we will use the set of probabilities $p(\text{useEdge}(1, 3)) = 0.2$, $p(\text{useEdge}(3, 4)) = 0.4$, and $p(\text{useEdge}(4, 5)) = 0.4$. It can be verified that this PASP instance is not satisfiable (intuitively, it's not possible to use edge (3, 4) without using edge (1, 3)).

Nevertheless, if we want to minimize the error of this instance we could make $\pi(I_1) = 0$, $\pi(I_2) = 0$, and $\pi(I_3) = 0.4$. As can be verified, this would leave us with $p(\text{useEdge}(1, 3)) = 0.4$, $p(\text{useEdge}(3, 4)) = 0.4$, and $p(\text{useEdge}(4, 5)) = 0.4$. The error of this answer, which equals the sum of the errors of each probability, is 0.2.

However, if we use Algorithm 1 the optimal value it will find is $p(\text{useEdge}(1, 3)) = 0.2$, $p(\text{useEdge}(3, 4)) = 0.2$, and $p(\text{useEdge}(4, 5)) = 0.2$, leaving us with an error of 0.4. That's because Algorithm 1 would never assign to an atom a value greater than it's associated probability.

The problem is that, since we search for a probability distribution over all subsets of the Herbrand base, HB_S , and no subset can have a negative probability, the probability of a literal is always lesser than or equal to the requested probability. If we only care about classifying an instance as either satisfiable or unsatisfiable, this is not important. However, if we seek to minimize the errors, it must be possible to sometimes assign a greater probability to a literal, as seen in example 5.

This calls for a change in the algorithm. In Linear Programming Problems if we want a variable x to be unconstrained, we define new variables x^+ and x^- and replace x for $x^+ - x^-$. Since we don't care about the probability of subsets that are not *Answer Sets* when looking for a approximated answer, the Linear Programming Problem we solve is:

$$\begin{aligned}
& \text{minimize} && c^T \pi \\
& [A|I] \pi = p \\
& \pi \geq 0 \\
& \sum \pi_i = 1
\end{aligned} \tag{13}$$

Where I is the Identity Matrix, each column of A is an *Answer Set*, and $c_i = 0$ if column i is an *Answer Set*, $c_i = 1$ otherwise.

In this system, the Identity Matrices function as error coefficients, or “slack variables” for each atom. If the value of π_i for a column i in the Identity Matrix is greater than zero, this value is summed or subtracted to the probability of the i -th atom. Therefore it is part of the error of this atom.

If the cost function reaches zero, we have a exact solution. Otherwise, we will have a solution with minimal error since the cost equals $E(\pi)$. The resulting π may not be a distribution probability over the subsets of HB_S , but a distribution with the *Answer Sets* have the same probability as π can be easily constructed.

6 Conclusion

In this work, we have defined the problem of Probabilistic Answer Set Programming, proposed an algorithm and a modification of this algorithm for finding approximated answers.

The search for appropriated answers to PASP problems is still a work in progress and may suffer several changes until we find a good solution. In this paper we show the current state of our researcher in the subject.

We hope this work will have several applications, including the implementation of a sophisticated part-of-speech tagger currently in progress that select tags to words in a way that approximate the relative frequency of linguistic phenomena in that particular tagging to those observed in native corpus.

We encourage other groups to try our PASP solver when facing problems with hard and soft constraints.

References

1. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. In: Lifschitz, V., Niemel, I. (eds.) LPNMR. Lecture Notes in Computer Science, vol. 2923, pp. 21–33. Springer (2004)
2. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. Theory Pract. Log. Program. 9(1), 57–144 (Jan 2009)
3. Bertsimas, D., Tsitsiklis, J.N.: Introduction to Linear Optimization. Athena Scientific, Belmont, Massachusetts, EUA (1997)
4. Boole, G.: An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities. Walton and Maberly (1854), <http://books.google.com/books?id=SWgLVT0otY8C>
5. Finger, M., Bras, R.L., Gomes, C., Selman, B.: Solutions for hard and soft constraints using optimized probabilistic satisfiability. Accepted for SAT2013 (2013)

6. Finger, M., De Bona, G.: A logic based algorithm for solving probabilistic satisfiability. In: Proceedings of the 12th Ibero-American conference on Advances in artificial intelligence. pp. 453–462. IBERAMIA'10, Springer-Verlag, Berlin, Heidelberg (2010)
7. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: International Conference on Logic Programming/Joint International Conference and Symposium on Logic Programming. pp. 1070–1080 (1988)
8. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: In The Logic Programming Paradigm: a 25-Year Perspective. pp. 375–398. Springer (1999)
9. de Morais, E.M.: Probabilistic Answer Set Programming (in Portuguese). Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo (2012)
10. de Morais, E.M., Finger, M.: Probabilistic answer set programming. In: Brazilian Conference on Intelligent Systems. pp. 150–156. Fortaleza, Brazil (2013)
11. Niemel, I., Simons, P., Soinen, T.: Stable model semantics of weight constraint rules. In: Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR99), Volume 1730 of lecture. pp. 317–331. Springer-Verlag. LNAI (1999)
12. Nilsson, N.J.: Probabilistic logic. *Artif. Intell.* 28(1), 71–87 (1986)
13. Prasolov, V., Tikhomirov, V.: *Geometry*. American Mathematical Society (julho 2001)
14. Van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. *J. ACM* 23(4), 733–742 (Oct 1976)
15. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters* 68(2), 63–69 (1998)
16. Weisstein, E.W.: Echelon form. From MathWorld—A Wolfram Web Resource (2012), <http://mathworld.wolfram.com/EchelonForm.html>