Implementing an Efficient SAT Solver for a Probabilistic Description Logic

Pavel Klinov
University of Manchester
Manchester, United Kingdom
pklinov@cs.man.ac.uk
and Bijan Parsia
University of Manchester
Manchester, United Kingdom
bparsia@cs.man.ac.uk

Abstract

This paper presents an optimized algorithm for solving the satisfiability problem (PSAT) in the probabilistic description logic P-$\mathcal{SROIQ}$. In P-$\mathcal{SROIQ}$ and related Nilsson-style probabilistic logics the PSAT problem is typically solved by reduction to linear programming. This straightforward approach does not scale well because the number of variables in linear programs grows exponentially with the number of probabilistic statements. In this paper we demonstrate an algorithm to cope with this problem which is based on column generation. Although column generation approaches to PSAT have been known for the last two decades, this is, to the best of our knowledge, the first algorithm which also works for a non-propositional probabilistic logic. We report results of an empirical investigation which show that the algorithm can handle probabilistic knowledge bases of about 1000 probabilistic statements in addition to even larger number of classical $\mathcal{SROIQ}$ axioms.

1 Introduction

There are many proposed formalisms for combining Description Logics (DLs) with various sorts of uncertainty, although, to our knowledge, none have been used in a real application. We believe that this is due to two reasons: 1) there is comparatively little knowledge about how to use these formalisms effectively (or even, which are best suited for what purposes) and 2) there is a severe lack of tooling, in particular, there have been no sufficiently effective reasoners.

This paper describes our work on the second problem. We present the satisfiability algorithm implemented in Pronto,[1] our reasoner for the probabilistic extension of DL $\mathcal{SROIQ}$ (named P-$\mathcal{SROIQ}$) [15]. This logic can be viewed either as a generalization of the Nilsson's propositional probabilistic logic [18] or as a fragment of first-order probabilistic logic of Halpern and Bacchus [5] [2] (with certain non-monotonic extensions which are unimportant in the context of this paper). One attractive feature of these probabilistic logics is that they allow modelers to declaratively describe their uncertain knowledge without fully specifying any probability distribution in contrast to, for example, Bayesian networks. They are also proper generalizations of their classical counterparts which, in the case of P-$\mathcal{SROIQ}$, means that modelers can take an existing $\mathcal{SROIQ}$ ontology and add probabilistic axioms to capture uncertain, such as statistical, relationships. Several applications of probabilistic DLs have been described, in particular, automated validation of uncertain ontology alignments [3] and, very recently, an analysis of a large medical expert system CADIAG-2 [14]. In the latter case uncertain rules used in medical diagnosis were translated into P-$\mathcal{SROIQ}$ and Pronto was used to discover *all* probabilistic inconsistencies in the system.

In spite of their attractive features Nilsson-style logics have been criticized, partly for the intractability of probabilistic inference. Reasoning procedures are typically implemented via

[1] http://www.cs.manchester.ac.uk/%7Eklinovp/research/pronto

reduction to linear programming but it is well known that corresponding linear programs are exponentially large so the scalability is very limited. Over the last two decades there have been several attempts to overcome that issue in the propositional case which led to some promising results, such as solving the probabilistic satisfiability problem (PSAT) for 800-1000 formulas [6]. It has been unclear whether the methods used to solve large propositional PSATs can be directly applied to PSAT in probabilistic DLs (see Section 5).

To the best of our knowledge, Pronto is the first reasoner for a Nilsson-style probabilistic DL which scalability is comparable to (and often better than) scalability of propositional solvers. In particular, it can solve propositional PSATs of the same size but i) can also handle probabilistic statements over arbitrary (i.e. non-propositional) $\mathcal{SROIQ}$ expressions and ii) can efficiently deal with KBs containing large bodies of non-probabilistic knowledge *in addition to* roughly 1000 probabilistic statements.

2 Preliminaries

This section provides a brief background on the Description Logic $\mathcal{SROIQ}$ and its probabilistic extension P-$\mathcal{SROIQ}$.

2.1 Description Logic

Description Logics is a family of logics which are typically decidable fragments of first-order logic developed specifically for representing structural background knowledge [1]. $\mathcal{SROIQ}$ is one of the most expressive representatives of that family. It is a formal basis of the Web Ontology Language (OWL 2) which is a W3C standard for representing ontologies. We introduce DLs using $\mathcal{ALCOQ}$— a subset of $\mathcal{SROIQ}$ which provides all the syntactic features used in this paper. Full presentation of $\mathcal{SROIQ}$ is space consuming and is not required since there are no important differences between probabilistic extensions to $\mathcal{ALCOQ}$ and to $\mathcal{SROIQ}$. We refer to [1, 9] for full details on syntax and semantics of expressive DLs including $\mathcal{SROIQ}$.

Syntax of $\mathcal{ALCOQ}$ We assume fixed finite sets N_C, N_R and N_I of concept names (atomic concepts), role names and individuals respectively. Concept expressions (or concepts) in $\mathcal{ALCOQ}$ have the following syntactic form:

$$C ::= A|\{o\}|\neg C|C \sqcap D|\exists R.C|\geq nR.C|\leq nR.C \tag{1}$$

where $A \in N_C, R \in N_R$, C and D are concepts, n is a natural number, $o \in N_I$. The following are the standard abbreviations: $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$, $\forall R.C \equiv \neg\exists R.\neg C$, $\bot \equiv \neg A \sqcap A$, $\top \equiv \neg\bot$, $\{o_1, \ldots, o_k\} \equiv \{o_1\}\sqcup, \ldots, \sqcup\{o_k\}$ and $= nR.C \equiv \geq nR.C \sqcap \leq nR.C$. Expressions of the form $\{o_1, \ldots, o_k\}$ are called nominals.

Usually a knowledge base (or *ontology*) in DLs is considered to be a tuple $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{R})$ where $\mathcal{T}$ is a terminological box (TBox), $\mathcal{A}$ is an assertional box (ABox) and $\mathcal{R}$ is a role box (RBox). In this paper RBoxes are irrelevant while nominals make TBoxes strictly more expressive than ABoxes [1]. Therefore we will only consider TBoxes which are sets of *concept subsumption* axioms. Each subsumption axiom is an expression of the form $C \sqsubseteq D$ where C and D are concepts. $C \equiv D$ abbreviates $\{C \sqsubseteq D, D \sqsubseteq C\}$.

Semantics of $\mathcal{ALCOQ}$ Semantics of DLs is standardly based on interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set (the domain) and $\cdot^{\mathcal{I}}$ is an interpretation function that maps each $A \in N_C$ to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each $R \in N_R$ to a relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and each $o \in N_I$ to an element $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. It is extended to concept expressions as follows:

$$
\begin{aligned}
&(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
&(\{o\})^{\mathcal{I}} = o^{\mathcal{I}} \\
&(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
&(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \exists y \in \mathcal{C}^{\mathcal{I}} \text{ s.t. } (x,y) \in R^{\mathcal{I}}\} \\
&(\geq nR.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} || \{y \in \mathcal{C}^{\mathcal{I}} \text{ s.t. } (x,y) \in R^{\mathcal{I}}\}| \geq n\} \\
&(\leq nR.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} || \{y \in \mathcal{C}^{\mathcal{I}} \text{ s.t. } (x,y) \in R^{\mathcal{I}}\}| \leq n\}
\end{aligned}
$$

An interpretation $\mathcal{I}$ satisfies (or is a model of) a subsumption axiom $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. It is a model of a TBox $\mathcal{T}$ if it is a model of each axiom in $\mathcal{T}$. A TBox is called satisfiable if it has a model. Given a TBox $\mathcal{T}$ concept C is called satisfiable if there exists an interpretation $\mathcal{I}$ which is a model of $\mathcal{T}$ and $C^{\mathcal{I}} \neq \emptyset$. An axiom $C \sqsubseteq D$ is entailed by a TBox $\mathcal{T}$ if it is satisfied by every model of $\mathcal{T}$.

2.2 Probabilistic Description Logic P-$\mathcal{SROIQ}$

P-$\mathcal{SROIQ}$ is a probabilistic generalization of the DL $\mathcal{SROIQ}$ [15]. It supports probabilistic subsumptions between arbitrary $\mathcal{SROIQ}$ concepts and a certain class of probabilistic concept assertions. Any $\mathcal{SROIQ}$ ontology can be used as a basis for a P-$\mathcal{SROIQ}$ ontology which facilitates transition from classical to probabilistic ontologies.

Syntax of P-$\mathcal{SROIQ}$ The syntactic constructs of P-$\mathcal{SROIQ}$ include those of $\mathcal{SROIQ}$ together with *conditional constraints*. Conditional constraints are expressions of the form $(D|C)[l,u]$ where D, C are $\mathcal{SROIQ}$ concept expressions (called *conclusion* and *evidence* respectively) and $[l,u] \subseteq [0,1]$ is a closed real-valued interval. Unconditional constraints are the special case of conditional ones when the evidence class is equivalent to $\top$.

For the purpose of this paper it is sufficient to consider only probabilistic TBoxes (or PTBoxes). A PTBox is a pair $PT = (\mathcal{T}, \mathcal{P})$ where $\mathcal{T}$ is a classical $\mathcal{SROIQ}$ TBox and P is a finite set of conditional constraints. Informally, a PTBox axiom $(D|C)[l,u]$ means that "if a randomly chosen individual is an instance of C, the probability of it being an instance of D is in $[l,u]$". In what follows we call $\mathcal{T}$ and $\mathcal{P}$ the classical and the probabilistic part of a PTBox respectively.

Semantics of P-$\mathcal{SROIQ}$ Semantics of P-$\mathcal{SROIQ}$ is standardly explained using the notion of *possible world* which is defined with respect to a set of concepts Φ (called *basic concepts* or *probabilistic signature*[2]) [15]. A possible world I is a subset of Φ such that the set of axioms $\{\{o\} \sqsubseteq C | C \in I\} \cup \{\{o\} \sqsubseteq \neg C | C \notin I\}$ is satisfiable for a fresh individual o (in other words, possible worlds correspond to *realizable* concept types). A basic concept C occurs *positively* in a possible world I if $C \in I$, otherwise it occurs *negatively*. The set of all possible worlds with respect to Φ is denoted as $\mathcal{I}_\Phi$. A world I satisfies a basic concept C denoted as $I \models C$ if C

[2]Note that basic concepts need not be atomic.

occurs positively in I. Satisfiability of basic concepts is inductively extended to $\mathcal{SROIQ}$ concept expressions according to the semantics of $\mathcal{SROIQ}$ (see Section 2.1), for example, $I \models C \sqcap D$ if $I \models C$ and $I \models D$.

For the reason which will become clear in Section 3.2 we assume a *linear order* of basic concepts in Φ. Since Φ is a finite set we can denote the i-th basic concept in Φ by C_i. For a given possible world I we also use the notation I_i to denote either C_i if C_i occurs positively in I or $\neg C_i$ if it occurs negatively. For a given PTBox the order of basic concepts is fixed across all possible worlds.

A world I is said to be a model of a TBox axiom α denoted as $I \models \alpha$ if $\alpha \cup \{\{o\} \sqsubseteq C | C \in I\} \cup \{\{o\} \sqsubseteq \neg C | C \notin I\}$ is satisfiable for a new individual o. A world I is a model of a $\mathcal{SROIQ}$ TBox $\mathcal{T}$ denoted as $I \models \mathcal{T}$ if it is a model of all axioms of $\mathcal{T}$. A world I that satisfies a TBox $\mathcal{T}$ exists iff $\mathcal{T}$ has a model $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ [15].

A probabilistic interpretation Pr is a probability distribution over $\mathcal{I}_\Phi$. Pr is said to *satisfy* a $\mathcal{SROIQ}$ TBox $\mathcal{T}$ denoted as $Pr \models \mathcal{T}$ if for all $I \in \mathcal{I}_\Phi, Pr(I) > 0$ implies that $I \models \mathcal{T}$. The probability of a concept C, denoted as $Pr(C)$, is defined as $\sum_{I \models C} Pr(I)$. $Pr(D|C)$ is used as an abbreviation for $Pr(C \sqcap D)/Pr(C)$ given $Pr(C) > 0$. A probabilistic interpretation Pr satisfies a conditional constraint $(D|C)[l, u]$, denoted as $Pr \models (D|C)[l, u]$, if $Pr(C) = 0$ or $Pr(D|C) \in [l, u]$. Pr satisfies a set of conditional constraints F if it satisfies each of the constraints. A PTBox $PT = (\mathcal{T}, \mathcal{P})$ is called *satisfiable* if there exists an interpretation that satisfies both $\mathcal{T}$ and $\mathcal{P}$.

The probabilistic satisfiability problem (PSAT) is a problem of deciding if a PTBox $(\mathcal{T}, \mathcal{P})$ has a model Pr. It is decidable and its complexity class is `N2ExpTime`-complete, i.e. the same as the complexity of reasoning in $\mathcal{SROIQ}$ [10]. We refer to [15] for a more detailed presentation of P-$\mathcal{SROIQ}$ semantics, reasoning problems and procedures, and complexity results.

3 The Probabilistic Satisfiability Algorithm

The first contribution of this paper is the novel PSAT algorithm implemented in Pronto (see Section 5 for some relationships to the previously developed methods). For the sake of clarity we will consider a special case of PSAT where the PTBox is of the form $PT = (\mathcal{T}, \{(C_i|\top)[p_i, p_i]\})$ (i.e. all probabilistic statements are unconditional constraints with point-valued probabilities and all C_i are concept names). It is straightforward, but technically awkward, to generalize the procedure to handle conditional interval statements over arbitrary concept expressions.

A PTBox $PT = (\mathcal{T}, \{(C_i|\top)[p_i, p_i]\})$ is satisfiable iff the following system of linear inequalities is *feasible*, i.e. admits at least one solution (by generalization from propositional PSAT [6]):

$$\begin{aligned} &\sum_{I \models C_i} x_I = p_i, \text{ for each } (C_i|\top)[p_i, p_i] \in \mathcal{P} \\ &\sum_{I \in \mathcal{I}_\Phi} x_I = 1 \text{ and all } x_I \geq 0 \end{aligned} \tag{2}$$

where $\mathcal{I}_\Phi$ is the set of all possible worlds for the set of concepts Φ in $\mathcal{T}$. Observe, that $\mathcal{I}_\Phi$ is finite but exponential in the size of Φ, so it is not feasible to explicitly generate this system to check whether it admits a solution.

One successful approach to dealing with linear systems having an exponential number variables is *column generation*. It is based on the fundamental property of linear programming:

any feasible (i.e. admitting at least one solution) program always has an optimal solution in which only a linear number of variables have non-zero values. Column generation exploits this property by trying to avoid an explicit representation of variables (columns) which will not have positive values in the finally discovered solution. The method is briefly presented in the next subsection.

3.1 Column Generation Basics

Consider the standard form of a linear program (3). Any linear program, in particular, a version (2) with intervals can be reduced to it by adding slack variables.

$$\max\ z = cx \tag{3}$$
$$\text{s.t. } Ax = b \tag{4}$$
$$x \geq 0$$

A denotes a $m \times n$ matrix of linear coefficients of (3). At every step of the simplex algorithm, A is represented as a combination (B, N) where B and N are submatrices of the *basic* and *non-basic* variables, respectively. Values of non-basic variables are fixed to zero, and the solver proceeds by replacing one basic variable by a non-basic one until the optimal solution is found. Variables are represented as indexed columns of A. The index of a non-basic column which enters the basis is determined according to the following expression [6]:

$$j \in \{1, \ldots, |N|\} \text{ s.t. } c_j - u^T A^j \text{ is maximal} \tag{5}$$

where c_j is the objective coefficient for the new variable and u^T is the current dual solution of (3). The expression $c_j - u^T A^j$ is called *reduced cost*. At every iteration the column having the highest positive reduced cost is selected. If no such column exists the linear program is at an optimum and the simplex algorithm stops.

If the size of N is exponential, as is the case for the program (2), one should compute the index of the entering column according to (5) without examining all columns in N. This is done using the column generation technique in which (5) is treated as an optimization problem with the following objective function:

$$max\ (c_j - \sum_{i=1}^{m+1} u_i a_i^j),\ A^j = (a_i^j) \in \{0,1\}^{m+1} \tag{6}$$

where a_i^j are binary variables that represent linear coefficients of the entering column.

It is important to note that except of the way the entering column is obtained (i.e. generated vs. selected) the simplex algorithm works along the same lines. Whether the column generation technique is successful is contingent upon the following criteria: i) there exists an efficient algorithm for the optimization problem (6), ii) an optimal solution of the program (3) can be found without generation of an excessive number of columns. Such number characterizes *convergence* of the algorithm. In the next subsection we demonstrate an optimized algorithm for generating columns for the PSAT system (2) and will later demonstrate its effectiveness.

3.2 Possible World Generation

We first rewrite the system (2) as the following linear program:

$$\begin{aligned} \max \ & \sum_{I \in I_\Phi} x_I \\ \text{s.t.} \ & \sum_{I \models C_i} x_I = p_i \times \sum_{I \in I_\Phi} x_I, \text{ for each } (C_i|\top)[p_i, p_i] \in \mathcal{P} \\ & \sum_{I \in I_\Phi} x_I \leq 1 \text{ and all } x_I \geq 0 \end{aligned} \tag{7}$$

This program has the optimal objective value of 1 iff the system (2) is feasible. The advantage of using this program is that it is feasible even if the PTBox is not satisfiable. This property facilitates use of the column generation technique.

Consider a_i^j, the i-th coefficient of some column A^j. The column corresponds to some possible world $I^j = \{C_i\}$, therefore $a_i^j = 1$ implies that C_i occurs positively in I^j while $a_i^j = 0$ implies that it occurs negatively (or equivalently, $I^j \models \neg C_i$). Thus it is possible to represent I^j as a *conjunctive* concept expression in $\mathcal{SROIQ}$ assuming a fixed linear ordering of concept names $\{C_i\}$ in Φ (see Section 2.2). More formally, we define the following function η which maps columns, i.e. binary vectors, to conjunctions of basic concepts from Φ:

$$\eta(A^j) = \bigsqcap X_i, \text{ where } X_i = \begin{cases} C_i, & \text{if } a_i^j = 1 \\ \neg C_i, & \text{if } a_i^j = 0 \end{cases} \tag{8}$$

X_i are literal concepts that denote either a basic concept or its negation.

Soundness of the PSAT algorithm strongly depends on whether every solution of the optimization problem (6), which is added as a column to the main linear program (7), corresponds to a concept expression that is satisfiable w.r.t. $\mathcal{T}$, i.e. is a *possible* world. If this condition is true then soundness trivially follows because one may simply enumerate the set of all solutions (since the set of possible worlds is finite), so (7) will be equivalent to the original linear system (2). *Completeness* requires that every possible world for the given PTBox corresponds to some solution of (6). Therefore, for ensuring both soundness and completeness it is crucial to construct a set of constraints $\mathcal{H}$ for the problem (6) such that its set of solutions is in one-to-one correspondence with the set of all possible worlds $\mathcal{I}_\Phi$.

In what follows we will call columns which correspond to satisfiable expressions *valid* and others — invalid. More formally, given a $\mathcal{SROIQ}$ TBox $\mathcal{T}$, a column A^j is valid if $\mathcal{T} \not\models \eta(A^j) \sqsubseteq \bot$ and is invalid otherwise.

Validity can easily be ensured in the propositional case where each C_i is a clause. One possibility is to employ a well known formulation of SAT as a mixed-integer linear program (MILP) [7]. For example, if $C_i = c_1 \vee \neg c_2 \vee c_3$ then (6) will have the constraint $a_i = x_{c1} + (1 - x_{c2}) + x_{c3}$ where all variables are binary. In that case soundness and completeness follow from the reduction of SAT to MILP. Previously developed propositional PSAT algorithms take full advantage of that (see Section 5).

In the case of an expressive language, such as $\mathcal{SROIQ}$, there appears to be no easy way of determining the set of constraints $\mathcal{H}$. Furthermore, it is unclear whether such a set is polynomial in the size of $\mathcal{T}$. Informally, $\mathcal{H}$ must capture every entailment, such as $\mathcal{T} \models C_i \sqcap, \ldots, \sqcap C_j \sqsubseteq \bot$ in order to prevent generation of any column A^j such that $C_i \sqcap, \ldots, \sqcap C_j$ is

a conjunctive subexpression of $\eta(A^j)$. All such entailments can be computed in a naive way by checking satisfiability of all conjunctions $C_i \sqcap, \ldots, \sqcap C_j$ over Φ but this is no better than trying to construct the full linear system (2).

Instead, Pronto implements a novel *hybrid*, *iterative* procedure to compute $\mathcal{H}$ which can be summarized as follows:

```
   Input: PTBox PT = (T, P), current dual solution u^T of (7)
   Output: New column A^j or null
 1 Initialize (6) using u^T, H ← ∅
 2 while A^j ≠ null do
 3 |   Solve (6) to optimality, A^j ← next optimal solution
 4 |   if A^j ≠ null then
 5 |   |   if satisfiable(η(A^j), T) then
 6 |   |   |   return A^j
 7 |   |   else
 8 |   |   |   add constraints to H that block A^j
 9 |   |   end
10 |   end
11 end
12 return null;
```

Algorithm 1: Possible world generation algorithm

The key steps are 5 and 8. On step 5 the algorithm invokes a $\mathcal{SROIQ}$ reasoner (in our case, Pellet [21]) to determine if the computed column corresponds to a *possible* world. This is critical for soundness. If yes, the column is valid and returned. If no, the current set of constraints $\mathcal{H}$ needs to be extended to exclude A^j from the set of solutions to (6) (a similar technique is used in All-SAT solvers to block clauses [17]). This step deserves a more detailed explanation which we present by first defining the notion of the minimal unsatisfiable core for an unsatisfiable conjunctive concept expression.

Definition 1 (Unsatisfiable Core)**.** *Given a TBox $\mathcal{T}$ and unsatisfiable (w.r.t. $\mathcal{T}$) concept expression $\bigsqcap X_i$ represented as a set of conjuncts $X = \{X_i\}$, a minimal unsatisfiable subexpression (MUS) is a subset $X' = \{X'_i\} \subseteq \{X_i\}$ such that $\bigsqcap X'_i$ is unsatisfiable w.r.t. $\mathcal{T}$ and any $X'' = \{X''_i\} \subset \{X'_i\}$ is satisfiable w.r.t. $\mathcal{T}$. The unsatisfiable core (UC) of $\bigsqcap X_i$ is the set of all its MUSes.*

Intuitively, our notion of UC for conjunctive $\mathcal{SROIQ}$ concepts corresponds to the standard notion of unsatisfiable core for propositional formulas in conjunctive normal form [16].

Each MUS can be regarded as a one "laconic justification" of the unsatisfiability of the original concept expression [8] (here "laconic" means that it contains no superfluous conjuncts). The UC is the set of all laconic justifications of the unsatisfiability. Clearly, it is sufficient to add a constraint that rules out *any* of the MUSes to exclude the current column from the set of solutions to (6).

Next, we show how to translate MUSes into linear inequalities. A MUS is a set of conjuncts $\{X'_i\}$ each of which corresponds to a binary variable (observe that η, as defined in (8), is a bijective function). By a slight abuse of notation we write $a_i = \eta^{-1}(X'_i)$ to denote the variable that corresponds to C_i. Then given a MUS $X' = \{X'_i\}_{i=1}^k$ we add the following linear constraint:

$$\sum_{i=1}^{k} a_i \leq k - 1, \text{ where } a_i = \begin{cases} \eta^{-1}(X'_i), & X'_i = C_i \\ 1 - \eta^{-1}(X'_i), & X_i = \neg C_i \end{cases} \tag{9}$$

If a conjunctive concept contains $\bigsqcap X_i$ as a subexpression then all binary expressions b_i, i.e. either a_i or $1 - a_i$ depending on whether X_i is a positive or a negative literal, are equal to 1. Therefore $\sum_{i=1}^{k} a_i = k$ where k is the size of $\{X_i\}$. Constraining $\sum_{i=1}^{k} b_i$ to be less or equal to $k - 1$ is equivalent to requiring at least one b_i to be equal to 0. According to the definition of η this is equivalent to removing of at least one conjunct from X' which makes it satisfiable (due to minimality of X', see Definition 1). Therefore, each of the constraints (9) is sufficient to exclude all columns, which correspond to concept expressions containing X', from the set of solutions to (6). Observe that the constraints do not exclude any columns which do *not* include X' since it is necessary to ensure completeness.

On step 8 the algorithm computes the unsatisfiable core for a concept expressions that corresponds to the current solution of (6). Then it transforms each of the MUSes into a linear inequality according to (9) and adds them to the binary program (6).

We call our PSAT algorithm (Algorithm 1) "hybrid" because it combines invocations of LP solver (to optimize (7)), MILP and $\mathcal{SROIQ}$ solvers (to optimize (6) and check satisfiability of concept expressions respectively). It is *iterative* because it iteratively tightens the set of solutions to (6) until either a valid column is found or provably no such column exists.

Finally, we give a short example demonstrating our iterative technique for computing valid columns.

Example 1. *Consider a PTBox where* $\mathcal{T} = \{A \sqsubseteq \exists R.C, B \sqsubseteq \exists R.\neg C, \geq 2R.\top \sqsubseteq D\}$ *and* $\mathcal{P}$ *contains some probabilistic constraints over the ordered set* $\Phi = \{A, B, D\}$. *Algorithm 1 starts out with an empty set of linear constraints for (6). The list of binary variables for (6) is* (x_A, x_B, x_D). *Assume that at some iteration the algorithm generates the following column:* $A^j = (1, 1, 0, 1)$ *(the last component of any column is always equal to* 1 *because all coefficients in the last row of (7) are equal to* 1*). Then* $\eta(A^j) = A \sqcap B \sqcap \neg D$.

It is not hard to see that $\mathcal{T} \models \eta(A^j) \sqsubseteq \bot$. *The reason is that any instance* o *of* $A \sqcap B$ *must have two R-successors (domain elements which are connected to* $o^{\mathcal{I}}$ *by* $R^{\mathcal{I}}$*). Moreover, they are necessarily distinct because one is an instance of* C *and another is an instance of* $\neg C$. *Therefore,* o *is an instance of* $\geq 2R.\top$ *and consequently is an instance of* D. *This is a contradiction with* $\neg D$ *in* $\eta(A^j)$.

The unsatisfiable core of $\eta(A^j)$ *is* $\{A, B, \neg D\}$. *This MUS is converted into the following linear inequality* $x_D \geq x_A + x_B - 1$ *which is then added to the binary program (6). As a result, no invalid column containing this MUS will be computed on subsequent iterations.*

3.3 Main Optimizations

We next describe several optimization techniques which play key roles for our implementation of the possible world generation algorithm.

3.3.1 Exploiting Concept Hierarchy

The first optimization stems from a natural observation that many inequalities for the binary program (6) can be added simply by examining the structure of a TBox. Virtually all modern DL reasoners can efficiently construct a so called *classification hierarchy* by finding all subsumptions between concept names that are logically entailed by the TBox. Such hierarchy can be used to construct an initial set of inequalities $\mathcal{H}_0$.

Consider the following TBox $\mathcal{T} = \{A \sqcup B \sqsubseteq C\}$. The classified version of $\mathcal{T}$ should include subsumptions $A \sqsubseteq C$ and $B \sqsubseteq C$. Therefore they can be directly translated to inequalities

$x_A \leq x_C$ and $x_B \leq x_C$ before computing some invalid column containing either $A \sqcap \neg C$ or $B \sqcap \neg C$ as subexpressions and converting these subexpressions into inequalities.

This idea helps to reduce the number of concept satisfiability tests. The effectiveness of this technique depends on axiomatic richness of the TBox. For axiomatically weak TBoxes, where almost all subsumptions can be discovered by traversing the concept hierarchy, most of the set $\mathcal{H}$ is computed up front. More complex TBoxes may have non-trivial entailments involving concept expressions on both left-hand and right-hand sides which can only be discovered when checking validity of some column candidate. One such example is the subsumption $A \sqcap B \sqsubseteq D$ from Example 1.

3.3.2 Optimistic Inequality Generation

One issue with a naive implementation of Algorithm 1 is that computing unsatisfiability cores may appear impractical for certain concept expressions and TBoxes. This may especially happen for long expressions which contain MUSes with little or no overlap. It is well known from the model diagnosis theory that finding all minimal unsatisfiable sets may require an exponential (in the size of all unsatisfiable sets) number of SAT tests [20].

To address this issue the algorithm imposes a time limit on the procedure that computes the UC. If at least one MUS has been found but finding others exceeds the timeout the procedure is interrupted. The found MUSes are then converted to linear inequalities and the algorithm moves on as if the full UC was computed.

This optimization does not cause a loss of either soundness or completeness. Completeness is trivially preserved because not adding some inequalities to the program (6) can only expand its set of solutions, so no possible world could be missed. Soundness is preserved because each computed column is still valid (SAT tests are never interrupted). The only possible negative impact of missing some MUSes is that they can appear in some future column candidates, so the algorithm might go through additional iterations. However, they do not *have to* appear because the optimal basis for the main program (7) can often be found before considering column candidates containing those MUSes. Intuitively, the algorithm behaves *optimistically* by hoping that additional iterations will not be required.

3.3.3 Multiple Column Generation and Stabilization

We use several techniques aimed at improving convergence of the column generation process. The most important are generating several optimal solutions of (6), i.e. column candidates, and introducing additional variables for the main linear program (7) to stabilize dual space and reduce degeneracy.

It is known that adding multiple columns into basis at every simplex iteration can improve convergence of column generation. For instance, Hansen and Perron add up to 50 column per iteration [6]. This is made possible by their heuristic method of generating columns (the variable neighborhood heuristics) which allows them to quickly generate many sub-optimal columns having positive reduced cost. In our case, when columns are generated by a MILP solver, there are the following two possibilities.

First, some MILP solvers, in particular, CPLEX, support *solution pools* which store multiple optimal or sub-optimal solutions for an MILP problem instance. Such solvers can either return the set of sub-optimal solutions which they recorded during the optimization process or continue the branch-and-cut search after the optimum until the required number of solutions has been found. Second, if the solver does not support solution pools one may still force it generate

multiple solutions. This can be done by "cutting off" the optimal solution and re-optimizing or by hooking inside the solver to continue the search after the optimum.

Pronto is highly modular and can be used with different LP/MILP solvers. Using a solution pool is the first choice strategy if it is available. If it is not available, as is the case with GLPK, then the second option is used. However, since it has implications for performance fewer columns are generated (usually up to 5).

Similarly to [6] we also use specific techniques to stabilize the values of the dual variables and reduce degeneracy.[3] This involves adding extra variables to the main linear program so it looks as follows:

$$\begin{aligned} \max \ & \sum_{I \in I_\Phi} x_I - \delta^+ y^+ - \delta^- y^- \\ \text{s.t.} \ & \sum_{I \models C_i} x_I = p_i + y^+ - y^- \\ & \sum_{I \in I_\Phi} x_I \leq 1, \ x_I, y^+, y^- \geq 0 \end{aligned} \tag{10}$$

where y^+ and y^- are the column vectors $(y_1^+, \ldots, y_m^+)^T$, $(y_1^-, \ldots, y_m^-)^T$ respectively while δ^+,δ^- are fixed positive coefficients. Observe that the original PSAT program (7) has the optimal value of 1 *iff* the augmented program has the optimal value of 1 (so soundness and completeness are preserved). However, the extra variables allow the objective function to vary smoothly between 0 and 1 which generally improves convergence of the column generation algorithms.

Our technique is simpler than the iterative method used by Hansen and Perron [6] but appears to be more efficient in most of our tests. At the same time one may combine different techniques or switch between them in the process of generating columns.

3.3.4 Early Unsatisfiability Detection

Automated reasoners often implement heuristic approaches to quickly detect obvious reasons for unsatisfiabilty of logical formulae. One good example of such techniques is the early clash detection methods employed by virtually all mature DL reasoners. Probabilistic reasoners are no exception, for instance, propositional PSAT solvers sometimes use incomplete rule-based reasoning methods which prove to be tremendously effective at proving unsatisfiability [6].

Since rule sets have only been formulated in the propositional case we take another approach. It is based on the observation that if the PSAT program has an optimal objective value of 1 (i.e. the KB is satisfiable) then the optimal dual solution is of the form $(0, \ldots, 0, 1)$. Components of the dual solution show the rate at which the objective value of the primal program will improve in response to a small relaxation of row bounds. If the optimal value of the program (7) is 1 then the only row whose relaxation can improve the objective is the last one, i.e. $\sum_{I \in I_\Phi} x_I \leq 1$. Its coefficients are exactly the same as the objective coefficients, thus its dual value is 1 while the other dual values are zeros. On the other hand, if the optimal objective value is below 1 then the indexes of non-zero dual variables indicate conflicting inequalities and, consequently, conflicting conditional constraints.

Our algorithm maintains history of dual solutions over a fixed number of column generation iterations. The history helps to spot the situation when the *same* dual variables repeatedly

[3] Degeneracy is the situation when some basic variables have zero values. It is known to be a problem for simplex since it can lead to multiple column swaps without an improvement of the objective value.

take on non-zero values while the primal objective is improving very slowly. More precisely, if over the last 10 iterations:

- The objective function improved by less than 1% and,
- More than 10% of dual values that were non-zero on *some* iterations had non-zero values on *all* iterations

then it is a good indication that the set of conditional constraints, which correspond to those dual variables, is not satisfiable. In that case, satisfiability of such constraints is tested separately. If they are indeed unsatisfiable, the whole process stops and unsatisfiability is reported, otherwise the main column generation loop continues.

The technique preserves *soundness* because of monotonicity: if some subset of a KB is unsatisfiable, then the whole KB is unsatisfiable. It also preserves *completeness* since for each satisfiable KB the column generation process will necessarily continue until satisfiability is proved (all extra tests, if any, must be negative).

The heuristic happens to be very effective for unsatisfiable KBs. In our experiments more than 90% of unsatisfiable KBs have been proved unsatisfiable by this method. It is effective mostly because conflicting sets of constraints tend to be small so it can easily be spotted if a small number of duals progress towards non-zero values. However, if the KB is satisfiable then the heuristic can slow the column generation down by introducing extra tests but this has so far happened in less than 15% of the cases.

4 Experimental Evaluation

Since P-$\mathcal{SROIQ}$ was designed as an extension of the DLs behind OWL and compatibility with OWL is declared as one of its major advantages, it is critical to ensure that the reasoning algorithms can successfully deal with probabilistic extensions of real OWL ontologies, not just randomly generated probabilistic clauses (as in [6]) or propositional taxonomies. We used the the following criteria to choose ontologies from a large variety of currently available ones:

- One of our long-term goals is to provide tools for reasoning over probabilistic extensions of OWL ontologies, so it is reasonable to evaluate the performance on ontologies which use as many features of OWL 2 as possible. At the same time ontologies represented in a lightweight fragment of OWL 2, such as OWL EL (based on logic $\mathcal{EL}^{++}$), also need to be included, especially given that they are getting increasingly widespread in some important domains, such as medical informatics.
- The ontologies should have reasonably large TBoxes with at least few hundred concept subsumption, equivalence or disjointness axioms, and some object roles.
- The ontologies should have at least 500 concepts in the TBox to show that the reasoner can handle large probabilistic signatures.
- Ideally, the ontologies should be "in service", i.e. have been created for and be in use by real applications as opposed to educational or experimental purposes. In that case they are more likely to encompass common and useful modeling patterns.

We selected the following five ontologies from different domains. For all ontologies we use versions stored in the TONES repository.[4]

[4] http://owl.cs.manchester.ac.uk/repository/

The Subcellular Anatomy Ontology (SAO) is the ontology from the neuroinformatics domain describing cellular and subcellular structures, supracellular domains, and macromolecules.[5] It contains 737 concepts, 915 subsumption, 4 equivalence, and 1580 concept disjointness axioms, 36 object properties and 47 data properties.

The Process Ontology is a part of the SWEET (Semantic Web for Earth and Environmental Terminology) collection of ontologies developed by NASA to provide semantic support for various Earth science projects.[6] It contains 1537 concepts, 1922 subsumption, 84 equivalence, and 1 concept disjointness axioms, 102 object properties and 19 data properties.

The Sequence Ontology with Composite Terms (SO-XP) defines terms and relationships used to describe features and attributes of biological sequence as well as cross-product definitions for composite terms.[7] It is a deliverable of the Gene Ontology Project and the Open Biomedical Ontologies (OBO) experiment. It contains 1660 concepts, 1709 subsumption, 198 equivalence, and 21 concept disjointness axioms and 22 object properties.

The Teleost Anatomy Ontology (TAO) is a multi-species anatomy ontology for teleost fishes.[8] It contains 2229 concepts, 3 object properties and 3406 concept subsumption axioms.

The Cell Type Ontology (CO) is a structured controlled vocabulary for cell types constructed for model organism and other Bioinformatics databases.[9] It contains 857 concepts, 1 object property and 1263 concept subsumption axioms.

Neither of these ontologies are propositional or small and simple enough to consider their propositionalization and a subsequent use of a propositional probabilistic SAT solver as a feasible alternative. None of the previously developed PSAT algorithms is capable of dealing with thousands of classical axioms in addition to a comparable number of probabilistic formulas.

PSAT instances over these ontologies were generated by the random selection of pairs of concept names. The proportion of unconditional constraints was kept at approximately 10%. Although there is not yet an established best practice for modeling in P-$\mathcal{SROIQ}$, our initial investigation [11] suggested that most PTBox constraints are conditional. This is also the case with Bayesian modeling. Unconditional constraints are mostly statements about individuals but since they are restricted (see [13]) their number is usually limited.

It is known that satisfiable probabilistic KBs are typically more difficult for PSAT algorithms [6, 4]. We have also observed that on average our algorithm generates less than half the columns for unsatisfiable instances as compared to satisfiable instances. This is mostly due to the early unsatisfiability detection technique (see Section 3.3.4). Therefore we implemented a specific technique to generate *satisfiable* KBs which follows the generic methodology used in [6]. It is based on generating two sets of possible worlds of fixed size with two probability distributions. Possible worlds are generated using a $\mathcal{SROIQ}$ reasoner such that each concept in the probabilistic signature has an approximately equal chance of being selected for the next world. Two probability functions Pr_1 and Pr_2 are then randomly selected from the set of normal probability distributions over the set of possible worlds. For each constraint $(D|C)[l, u]$ we take l (resp. u) as the minimum (resp. maximum) probability which is assigned to $(D|C)$ by Pr_1 and Pr_2. Intuitively, the existence of these interpretations guarantees satisfiability of the KB because they are some of its models. The early unsatisfiability detection heuristic has been kept on during the experiments to account for any loss of performance it may cause.

[5] http://ccdb.ucsd.edu/CCDBWebSite/sao.html
[6] http://sweet.jpl.nasa.gov/ontology/
[7] http://wiki.geneontology.org/index.php/SO:Composite_Terms
[8] https://www.nescent.org/phenoscape/Teleost_Anatomy_Ontology
[9] http://obolibrary.org/cgi-bin/detail.cgi?id=cell

IBM CPLEX solver was used to solve all LP and MILP problem instances. Other solvers can be easily plugged into Pronto. We have experimented with GLPK[10] and the results tend to be 25%-50% worse than those presented below.

All the experiments were conducted on a desktop PC with 2GHz CPU and 2GB RAM with JRE 1.6 running under Windows XP SP3. For each PSAT test we measured the following parameters: total CPU time (in seconds), total number of columns generated, and the average time to generate a new column (CG time, in milliseconds). CPU time and the number columns are averaged over 10 PSAT instances for each KB's size while CG time is also averaged over all columns generated during a single PSAT. The results are presented in Table 1.

Table 1: Results of PSAT evaluation on satisfiable PTBoxes

Ontology	Language	TBox size	Signature size	PTBox size	Total time (s)	CG time (ms)	# columns
SAO	$\mathcal{SHIN}$	2499	125	250	78.85	449.86	138.8
			250	500	161.16	450.24	261
			325	750	360.97	1023.86	351.4
			500	1000	1275.85	3062.52	424.8
Process	$\mathcal{SHOF}$	2007	125	250	48.48	337.38	87.2
			250	500	114.68	380.76	180.6
			325	750	224.42	480.26	280.8
			500	1000	416.66	509.26	408.8
SO-XP	$\mathcal{SHI}$	1928	125	250	58.62	423.08	74.4
			250	500	186.49	741.76	184.8
			325	750	424.66	986.8	318.4
			500	1000	799.38	1524.4	422.4
TAO	$\mathcal{EL}^{++}$	3406	125	250	48.63	280.96	91.4
			250	500	123.36	379.68	191
			325	750	224.69	414.16	281.4
			500	1000	430.69	466.54	449.02
Cell Type	$\mathcal{EL}^{++}$	1263	125	250	53.8	321.1	87.2
			250	500	128.2	386.54	180.8
			325	750	236.85	428.4	280.8
			500	1000	420.93	484.78	394.6

The major outcome is that our algorithm exhibits a very good convergence on probabilistic extensions of real ontologies. This appears to be mostly due to rich TBoxes which effectively shrink the set of all possible world thus allowing the algorithm to faster arrive at the optimal PSAT program. For instance, in the case of the SAO ontology our algorithm was able to solve all instances of PSAT of size 1000 never generating more than 600 columns out of the space of 2^{500} (excluding invalid ones). A likely explanation is a high number of concept disjointness axioms in the SAO ontology.

Due to the space restriction we do not include the results on propositional KBs. In that case Pronto's performance is comparable to that of propositional PSAT algorithms [6, 4]. The algorithm tends to have a worse convergence on such KBs but columns are generated faster because the optimization program (6), that is used to produce columns, has fewer inequalities. This is a direct consequence of simpler TBoxes.

Our plan is to continue the experiments with probabilistic extensions of real ontologies. Apart from posing interesting challenges to the PSAT algorithm, such ontologies are also more valuable with respect to developing useful methodologies for probabilistic modeling. They

[10]The GNU Linear Programming Kit, http://www.gnu.org/software/glpk/

present examples of interesting interactions between classical and probabilistic knowledge which can be useful for augmenting real ontologies with, for example, domain statistics.

5 Related Work

There is extensive work on solving the PSAT problem in propositional probabilistic logic (see esp. [6, 19, 4, 7]). Those algorithms are similar in spirit since they are also based on the column generation technique but are different in scope, design, and implementation.

The major difference between propositional and non-propositional PSAT problems in the context of column generation is that propositionality of the KB allows encoding of all its structure in a polynomial number of linear inequalities over a polynomial number of binary variables. This follows directly from the well known reduction of propositional SAT to integer programming [7]. Therefore, the column generation problem (6) is much easier to handle, either as a standard MILP instance [7, 4] or as a non-linear 0-1 program [6, 19]. To the best of our knowledge, this work is the first to describe an algorithm and evaluation results for non-propositional PSAT.

Our method does *not* try to capture the classical part of the KB at once. By interacting with a classical SAT solver for the target logic we capture only those parts which are essential for solving a particular PSAT or entailment problem. While this may be less efficient on KBs with weak classical part (or expressed in a restricted language, such as propositional logic) it has important advantages. First, it allows us to handle essentially any target logic for which a SAT solver is available. Second, this makes our algorithm more scalable with respect to the amount of classical knowledge. For example, modern DL reasoners can efficiently solve concept satisfiability problems even for very large TBoxes containing thousands of axioms (a characteristic example is the NCI Thesaurus). Even if they were propositional (or could be propositionalized) and fully translated into constraints for the problem (6), the latter would be drastically larger. Instead, as our results show, we can often solve PSAT by capturing only some relevant parts of TBoxes.

The PSAT algorithm presented in this paper is a substantial improvement of the earlier described algorithm [12] which is based on a similar idea but generates columns by solving a generic constraint optimization problem. The MILP formulation, presented in this paper, appears to be much more tractable and amenable to optimizations which resulted in the scalability improvements of the order of magnitude.

6 Conclusion

The primary conclusion of this work is that it is highly likely that P-$\mathcal{SROIQ}$ is as "practical" an ontology language, at least from the computational point of view, as $\mathcal{SROIQ}$. PSAT and the various entailment problems for P-$\mathcal{SROIQ}$ are `N2ExpTime`-complete, so, as with any logic in the $\mathcal{SROIQ}$ family, practicality, efficiency, and scalability claims must be carefully qualified. However, our current system handles all our randomly generated problems, which stress characteristics that we have good reason to believe to be present in future ontologies, to scales that are reasonable for hand built probabilistic ontologies in the next few years. As with $\mathcal{SROIQ}$, it is certainly possible to generate — or to find naturally — KBs that will utterly defeat our current optimizations, but the transformation from being barely able to handle 10-50 probabilistic statements to being handle thousands is a game changing improvement. Of

course, $\mathcal{SROIQ}$ reasoners have a much larger suite of optimizations to cope with a broader set of ontology types. But this is a difference in degree, not kind.

For example, prior to the current PSAT algorithm the breast cancer risk ontology (BCRA) [11] was nigh impossible to work with. We were reduced to working only with subsets of the ontology and had to abandon the idea of importing it into a new ovarian cancer ontology. This is clearly an unacceptable development situation for probabilistic ontology modelers. But now it makes sense for ontology modelers to experiment seriously with P-$\mathcal{SROIQ}$: Not only is it likely that Pronto can handle their needs, but we now have confidence that additional optimizations (or workarounds) can be designed as we encounter new difficult problems. In this sense, we are now on the same footing as the $\mathcal{SROIQ}$ family of logics: Modelers keep breaking reasoners and reasoners keep adapting to the new challenges.

References

[1] F. Baader, D. Calvanese, D. McGuiness, D. Nardi, and P. F. Patel-Schneider. *Description Logic Handbook*. Cambridge University Press, 2003.

[2] F. Bacchus. *Representing and reasoning with probabilistic knowledge*. MIT Press, 1990.

[3] S. Castano, A. Ferrara, D. Lorusso, T. H. Näth, and R. Möller. Mapping validation by probabilistic reasoning. In *European Conference on Semantic Web*, pages 170–184, 2008.

[4] P. S. de Souza Andrade, J. C. F. da Rocha, D. P. Couto, A. da Costa Teves, and F. G. Cozman. A toolset for propositional probabilistic logic. In *Encontro Nacional de Inteligencia Artificial*, pages 1371–1380, 2007.

[5] J. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.

[6] P. Hansen and S. Perron. Merging the local and global approaches to probabilistic satisfiability. *Int. Journal of Approximate Reasoning*, 47(2):125–140, 2008.

[7] J. Hooker. Quantitative approach to logical reasoning. *Decision Support Systems*, 4:45–69, 1988.

[8] M. Horridge, B. Parsia, and U. Sattler. Laconic and precise justifications in OWL. In *International Semantic Web Conference*, pages 323–338, 2008.

[9] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *Knowledge Representation and Reasoning*, pages 57–67, 2006.

[10] Y. Kazakov. SRIQ and SROIQ are harder than SHOIQ. In *Knowledge Representation and Reasoning*, pages 274–284, 2008.

[11] P. Klinov and B. Parsia. Probabilistic modeling and OWL: A user oriented introduction into P-$\mathcal{SHIQ}$(D). In *OWL: Experiences and Directions (OWLED-2008)*, 2008.

[12] P. Klinov and B. Parsia. On improving the scalability of checking satisfiability in probabilistic description logics. In *International Conference on Scalable Uncertainty Management*, volume 5785/2009 of *Lecture Notes in Computer Science*, pages 138–149. Springer, 2009.

[13] P. Klinov and B. Parsia. Relationships between probabilistic description and first-order logics. In *International Workshop on Uncertainty in Description Logics*, 2010.

[14] P. Klinov, B. Parsia, and D. Picado. The consistency of the CADIAG-2 knowledge base: A probabilistic approach. In *Logic for Programming, Artificial Intelligence and Reasoning*, 2010.

[15] T. Lukasiewicz. Expressive probabilistic description logics. *Artificial Intelligence*, 172(6-7):852–883, 2008.

[16] I. Lynce and J. P. M. Silva. On computing minimum unsatisfiable cores. In *International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[17] K. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Computer Aided Verification*, pages 250–264, 2002.

[18] N. J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.

[19] Z. Ognjanovic, U. Midic, and N. Mladenovic. A hybrid genetic and variable neighborhood descent for probabilistic SAT problem. In *Hybrid Metaheuristics*, pages 42–53, 2005.

[20] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[21] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.