

# Function synthesis for maximizing model counting

Thomas Vigouroux<sup>[0000–0001–6396–0285]</sup>, Marius Bozga<sup>[0000–0003–4412–5684]</sup>,  
Cristian Ene<sup>[0000–0001–6322–0383]</sup>, and Laurent Mounier<sup>[0000–0001–9925–098X]</sup>

Univ. Grenoble Alpes, CNRS, Grenoble INP<sup>\*\*</sup>, VERIMAG, 38000 Grenoble, France  
{name.surname}@univ-grenoble-alpes.fr <https://www-verimag.imag.fr/>

**Abstract.** Given a boolean formula  $\phi(X, Y, Z)$ , the *Max#SAT* problem [10,26] asks for finding a partial model on the set of variables  $X$ , maximizing its number of projected models over the set of variables  $Y$ . We investigate a strict generalization of *Max#SAT* allowing dependencies for variables in  $X$ , effectively turning it into a synthesis problem. We show that this new problem, called *DQMax#SAT*, subsumes both the *DQBF* [22] and *DSSAT* [18] problems. We provide a general resolution method, based on a reduction to *Max#SAT*, together with two improvements for dealing with its inherent complexity. We further discuss a concrete application of *DQMax#SAT* for symbolic synthesis of adaptive attackers in the field of program security. Finally, we report preliminary results obtained on the resolution of benchmark problems using a prototype *DQMax#SAT* solver implementation.

**Keywords:** Function synthesis · Model counting · *Max#SAT* · *DQBF* · *DSSAT* · Adaptive attackers.

## 1 Introduction

A major concern in software security are active adversaries, i.e., adversaries that can *interact* with a target program by feeding inputs. Moreover, these adversaries can often make observations about the program execution through side-channels and/or legal outputs. In this paper, we consider *adaptive* adversaries, i.e., adversaries that choose their inputs by taking advantage of previous observations.

In order to get an upper bound of the insecurity of a given program with respect to this class of adversaries, a possible approach is to synthesize the *best* adaptive attack strategy. This can be modelled as finding a function  $A$  (corresponding to the adversarial strategy) satisfying some logical formula  $\Phi$  (capturing some combination of attack objectives). Actually, this corresponds to a classical functional synthesis problem.

Informally, in our case, given a Boolean relation  $\Phi$  between output variables (observables) and input variables (attacker provided), our goal is to synthesize each input variable as a function on preceeding outputs satisfying  $\Phi$ . In the literature, this synthesis problem is captured by the so-called Quantified Boolean

---

<sup>\*\*</sup> Institute of Engineering Univ. Grenoble Alpes

Formulae (*QBF*) satisfiability problem [11,12] and its generalization, the Dependency Quantified Boolean Formulae (*DQBF*) satisfiability problem [22].

These existing qualitative frameworks are not sufficient in a security context: we are not only interested by adversaries able to succeed *in all cases*, but rather for adversaries succeeding with “a good probability”. The Stochastic SAT (*SSAT*) problem [20] was therefore proposed and replaces the classical universal (resp. existential) quantifiers by *counting* (resp. *maximizing*) quantifiers. This corresponds to finding the optimal inputs, depending on preceeding outputs, that maximize the number of models of  $\Phi$ , hence the succeeding probability of the attack. More recently, the Dependency Stochastic SAT (*DSSAT*) problem [18] has been proposed as a strict generalization of the *SSAT* problem by allowing explicit dependencies for maximizing variables, in a similar way the *DQBF* problem generalizes the *QBF* problem.

Nonetheless, an additional complication is hindering the use of quantitative stochastic frameworks in our security context. In general, the output variables in a program may hold expressions computed from one or more secret variables. Consequently, they rarely translate as counting variables in a stochastic formula. Most likely, the above-mentioned secret variables translate into counting variables whereas the observable variables need to be projected out when counting the models. Yet, the output variables are mandatory to express the knowledge available and the dependencies for synthesizing the attacker’s optimal inputs.

As an example, we are interested in solving counting problems of the form:

$$\max^{\{z_1\}} x_1. \max^{\{z_2\}} x_2. \mathcal{R}y_1. \mathcal{R}y_2. \exists z_1. \exists z_2. \\ (x_1 \Rightarrow y_2) \wedge (y_1 \Rightarrow x_2) \wedge (y_1 \vee z_2 \Leftrightarrow y_2 \wedge z_1)$$

which involve three distinct types of quantified variables and which are interpreted as follows: synthesize for  $x_1$  (respectively  $x_2$ ) a boolean expression  $e_1$  (respectively  $e_2$ ), depending only on  $z_1$  (respectively  $z_2$ ), such that the formula obtained after replacing  $x_i$  by  $e_i$  has a maximal number of models *projected* on the counting variables  $y_1, y_2$ .

Notice that this problem generalizes in a non-trivial way three well-known existing problems: (i) it generalizes the *Max#SAT* problem [10,26] by allowing the maximizing variables to depend *symbolically* on other variables; (ii) it lifts the *DQBF* problem [22] to a quantitative problem, we do not want to check if there exist expressions  $e_i$  working for all  $y_1, y_2$ , but to find expressions  $e_i$  maximizing the number of models on  $y_1, y_2$ ; (iii) it extends the *DSSAT* problem [18] with the additional category of *existential* variables, which can occur in the dependencies of maximizing variables, but which are projected for model counting.

Our contributions are the following:

- We introduce formally the *DQMax#SAT* problem as a new problem that arises naturally in the field of software security, and we show that it subsumes the *Max#SAT*, *DQBF* and *DSSAT* problems.
- We develop a general resolution method based on a reduction to *Max#SAT* and further propose two improvements in order to deal with its inherent

- complexity: (i) an incremental method, that enables anytime resolution; (ii) a local method, allowing to split the initial problem into independent smaller sub-problems, enabling parallel resolution.
- We provide two applications of  $DQMax\#SAT$  to software security: we show that *quantitative robustness* [3] and *programs as information leakage-channels* [25,23] can be systematically cast as instances of the  $DQMax\#SAT$  problem.
- We provide a first working prototype solver for the  $DQMax\#SAT$  problem and we apply it to the examples considered in this paper.

The paper is organized as follows. Section 2 introduces formally the  $DQMax\#SAT$  problem and its relation with the  $Max\#SAT$ ,  $DQBF$  and  $DSSAT$  problems. Sections 3 to 5 present the three different approaches we propose for solving  $DQMax\#SAT$ . Section 6 shows concrete applications of  $DQMax\#SAT$  in software security, that is, for the synthesis of adaptive attackers. Finally, Section 7 provides preliminary experimental results obtained with our prototype  $DQMax\#SAT$  solver. Section 8 discusses some references to related work and Section 9 concludes and proposes some extensions to address in the future.

## 2 Problem statement

### 2.1 Preliminaries

Given a set  $V$  of Boolean variables, we denote by  $\mathcal{F}\langle V \rangle$  (resp.  $\mathcal{M}\langle V \rangle$ ) the set of Boolean formulae (resp. complete monomials) over  $V$ . A model of a boolean formula  $\phi \in \mathcal{F}\langle V \rangle$  is an assignement  $\alpha_V : V \rightarrow \mathbb{B}$  of variables to Boolean values such that  $\phi$  evaluates to  $\top$  (that is, *true*) on  $\alpha_V$ , it is denoted by  $\alpha_V \models \phi$ . A formula is satisfiable if it has at least one model  $\alpha_V$ . A formula is valid (i.e., tautology) if any assignement  $\alpha_V$  is a model.

Given a formula  $\phi \in \mathcal{F}\langle V \rangle$  we denote by  $|\phi|_V$  the number of its models, formally  $|\phi|_V \stackrel{def}{=} |\{\alpha_V : V \rightarrow \mathbb{B} \mid \alpha_V \models \phi\}|$ . For a partitioning  $V = V_1 \uplus V_2$  we denote by  $|\exists V_2. \phi|_{V_1}$  the number of its  $V_1$ -projected models, formally  $|\exists V_2. \phi|_{V_1} \stackrel{def}{=} |\{\alpha_{V_1} : V_1 \rightarrow \mathbb{B} \mid \exists \alpha_{V_2} : V_2 \rightarrow \mathbb{B}. \alpha_{V_1} \uplus \alpha_{V_2} \models \phi\}|$ . Note that in general  $|\exists V_2. \phi|_{V_1} \leq |\phi|_V$  with equality only in some restricted situations (e.g. when  $V_1$  is an independent support of the formula [6]).

Let  $V, V', V''$  be arbitrary sets of Boolean variables. Given a Boolean formula  $\phi \in \mathcal{F}\langle V \rangle$  and a substitution  $\sigma : V' \rightarrow \mathcal{F}\langle V'' \rangle$  we denote by  $\phi[\sigma]$  the Boolean formula in  $\mathcal{F}\langle (V \setminus V') \cup V'' \rangle$  obtained by replacing in  $\phi$  all occurrences of variables  $v'$  from  $V'$  by the associated formula  $\sigma(v')$ .

### 2.2 Problem Formulation

**Definition 1 ( $DQMax\#SAT$  problem).** Let  $X = \{x_1, \dots, x_n\}$ ,  $Y, Z$  be pairwise disjoint finite sets of Boolean variables, called respectively maximizing, counting and existential variables. The  $DQMax\#SAT$  problem is specified as:

$$\max^{H_1} x_1. \dots \max^{H_n} x_n. \Re Y. \exists Z. \Phi(X, Y, Z) \quad (1)$$

where  $H_1, \dots, H_n \subseteq Y \cup Z$  and  $\Phi \in \mathcal{F}\langle X \cup Y \cup Z \rangle$  are respectively the dependencies of maximizing variables and the objective formula.

The solution to the problem is a substitution  $\sigma_X^* : X \rightarrow \mathcal{F}\langle Y \cup Z \rangle$  associating formulae on counting and existential variables to maximizing variables such that (i)  $\sigma_X^*(x_i) \in \mathcal{F}\langle H_i \rangle$ , for all  $i \in [1, n]$  and (ii)  $|\exists Z. \Phi[\sigma_X^*]|_Y$  is maximal. That means, the chosen substitution conforms to dependencies on maximizing variables and guarantees the objective holds for the largest number of models projected on the counting variables.

*Example 1.* Consider the problem:

$$\max^{\{z_1, z_2\}} x_1. \mathcal{R}y_1. \mathcal{R}y_2. \exists z_1. \exists z_2. (x_1 \Leftrightarrow y_1) \wedge (z_1 \Leftrightarrow y_1 \vee y_2) \wedge (z_2 \Leftrightarrow y_1 \wedge y_2)$$

Let  $\Phi$  denote the objective formula. In this case,  $\mathcal{F}\langle\{z_1, z_2\}\rangle = \{\top, \perp, z_1, \overline{z_1}, z_2, \overline{z_2}, z_1 \vee z_2, \overline{z_1} \vee \overline{z_2}, z_1 \vee \overline{z_2}, \overline{z_1} \vee \overline{z_2}, z_1 \wedge z_2, \overline{z_1} \wedge \overline{z_2}, z_1 \wedge \overline{z_2}, \overline{z_1} \wedge \overline{z_2}, z_1 \Leftrightarrow z_2, \overline{z_1} \Leftrightarrow \overline{z_2}\}$ , and one shall consider every possible substitution. One can compute for instance  $\Phi[x_1 \mapsto \overline{z_1} \wedge \overline{z_2}] \equiv ((\overline{z_1} \wedge \overline{z_2}) \Leftrightarrow y_1) \wedge (z_1 \Leftrightarrow y_1 \vee y_2) \wedge (z_2 \Leftrightarrow y_1 \wedge y_2)$  which only has one model ( $\{y_1 \mapsto \perp, y_2 \mapsto \top, z_1 \mapsto \top, z_2 \mapsto \perp\}$ ) and henceforth  $|\exists z_1. \exists z_2. \Phi[x_1 \mapsto \overline{z_1} \wedge \overline{z_2}]|_{\{y_1, y_2\}} = 1$ . Overall, for this problem there exists four possible maximizing substitutions  $\sigma^*$  respectively  $x_1 \mapsto z_1, x_1 \mapsto z_2, x_1 \mapsto z_1 \vee z_2, x_1 \mapsto z_1 \wedge z_2$  such that for all of them  $|\exists z_1. \exists z_2. \Phi[\sigma^*]|_{\{y_1, y_2\}} = 3$ .

*Example 2.* Let us consider the following problem:

$$\max^{\{z_1\}} x_1. \max^{\{z_2\}} x_2. \mathcal{R}y_1. \mathcal{R}y_2. \exists z_1. \exists z_2. (x_1 \Rightarrow y_2) \wedge (y_1 \Rightarrow x_2) \wedge (y_1 \vee z_2 \Leftrightarrow y_2 \wedge z_1)$$

Let  $\Phi$  denote the associated objective formula. An optimal solution is  $x_1 \mapsto \perp, x_2 \mapsto \overline{z_2}$  and one can check that  $|\exists z_1. \exists z_2. \Phi[x_1 \mapsto \perp, x_2 \mapsto \overline{z_2}]|_{\{y_1, y_2\}} = 3$ . Moreover, one can notice that there do not exist expressions  $e_1 \in \mathcal{F}\langle\{z_1\}\rangle$  (respectively  $e_2 \in \mathcal{F}\langle\{z_2\}\rangle$ ), such that  $\exists z_1. \exists z_2. \Phi[x_1 \mapsto e_1, x_2 \mapsto e_2]$  admits the model  $y_1 \mapsto \top, y_2 \mapsto \perp$ .

The following proposition provides an upper bound on the number of models corresponding to the solution of (1) computable using projected model counting.

**Proposition 1.** *For any substitution  $\sigma_X : X \rightarrow \mathcal{F}\langle Y \cup Z \rangle$  it holds*

$$|\exists Z. \Phi[\sigma_X]|_Y \leq |\exists X. \exists Z. \Phi|_Y.$$

### 2.3 Hardness of $DQMax\#SAT$

We briefly discuss now the relationship between the  $DQMax\#SAT$  problem and the  $Max\#SAT$ ,  $DQBF$  and  $DSSAT$  problems. It turns out that  $DQMax\#SAT$  is at least as hard as all of them, as illustrated by the following reductions.

***DQMax#SAT* is at least as hard as *Max#SAT*:** Let  $X = \{x_1, \dots, x_n\}$ ,  $Y, Z$  be pairwise disjoint finite sets of Boolean variables, called *maximizing*, *counting* and *existential* variables. The *Max#SAT* problem [10] specified as

$$\max x_1. \dots \max x_n. \mathfrak{R}Y. \exists Z. \Phi(X, Y, Z) \quad (2)$$

asks for finding an assignment  $\alpha_X^* : X \rightarrow \mathbb{B}$  of maximizing variables to Boolean values such that  $|\exists Z. \Phi[\alpha_X^*]|_Y$  is maximal. It is immediate to see that the *Max#SAT* problem is the particular case of the *DQMax#SAT* problem where there are no dependencies, that is,  $H_1 = H_2 = \dots = H_n = \emptyset$ .

***DQMax#SAT* is at least as hard as *DQBF*:** Let  $X = \{x_1, \dots, x_n\}$ ,  $Y$  be disjoint finite sets of Boolean variables and let  $H_1, \dots, H_n \subseteq Y$ . The *DQBF* problem [22] asks, given a *DQBF* formula:

$$\forall Y. \exists^{H_1} x_1. \dots \exists^{H_n} x_n. \Phi(X, Y) \quad (3)$$

to synthesize a substitution  $\sigma_X^* : X \rightarrow \mathcal{F}\langle Y \rangle$  whenever one exists such that (i)  $\sigma_X^*(x_i) \in \mathcal{F}\langle H_i \rangle$ , for all  $i \in [1, n]$  and (ii)  $\Phi[\sigma_X^*]$  is valid. The *DQBF* problem is reduced to the *DQMax#SAT* problem:

$$\max^{H_1} x_1. \dots \max^{H_n} x_n. \mathfrak{R}Y. \Phi(X, Y) \quad (4)$$

By solving (4) one can solve the initial *DQBF* problem (3). Indeed, let  $\sigma_X^* : X \rightarrow \mathcal{F}\langle Y \rangle$  be a solution for (4). Then, the *DQBF* problem admits a solution if and only if  $|\Phi[\sigma_X^*]|_Y = 2^{|Y|}$ . Moreover,  $\sigma_X^*$  is a solution for the problem (3) because (i)  $\sigma_X^*$  satisfies dependencies and (ii)  $\Phi[\sigma_X^*]$  is valid as it belongs to  $\mathcal{F}\langle Y \rangle$  and has  $2^{|Y|}$  models. Note that through this reduction of *DQBF* to *DQMax#SAT*, the maximizing quantifiers in *DQMax#SAT* can be viewed as Henkin quantifiers [14] in *DQBF* with a quantitative flavor.

***DQMax#SAT* is at least as hard as *DSSAT*:** Let  $X = \{x_1, \dots, x_n\}$ ,  $Y = \{y_1, \dots, y_m\}$  be disjoint finite sets of variables. A *DSSAT* formula is of the form:

$$\max^{H_1} x_1. \dots \max^{H_n} x_n. \mathfrak{R}^{p_1} y_1. \dots \mathfrak{R}^{p_m} y_m. \Phi(X, Y) \quad (5)$$

where  $p_1, \dots, p_m \in [0, 1]$  are respectively the probabilities of variables  $y_1, \dots, y_m$  to be assigned  $\top$  and  $H_1, \dots, H_n \subseteq Y$  are respectively the dependency sets of variables  $x_1, \dots, x_n$ . Given a *DSSAT* formula (5), the probability of an assignment  $\alpha_Y : Y \rightarrow \mathbb{B}$  is defined as

$$\mathbb{P}[\alpha_Y] \stackrel{def}{=} \prod_{i=1}^m \begin{cases} p_i & \text{if } \alpha_Y(y_i) = \top \\ 1 - p_i & \text{if } \alpha_Y(y_i) = \perp \end{cases}$$

This definition is lifted to formula  $\Psi \in \mathcal{F}\langle Y \rangle$  by summing up the probabilities of its models, that is,  $\mathbb{P}[\Psi] \stackrel{def}{=} \sum_{\alpha_Y \models \Psi} \mathbb{P}[\alpha_Y]$ .

The *DSSAT* problem [18] asks, for a given formula (5), to synthesize a substitution  $\sigma_X^* : X \rightarrow \mathcal{F}\langle Y \rangle$  such that (i)  $\sigma_X^*(x_i) \in \mathcal{F}\langle H_i \rangle$ , for all  $i \in [1, n]$  and (ii)  $\mathbb{P}[\Phi[\sigma_X^*]]$  is maximal. If  $p_1 = \dots = p_m = \frac{1}{2}$  then for any substitution  $\sigma_X : X \rightarrow \mathcal{F}\langle Y \rangle$  it holds  $\mathbb{P}[\Phi[\sigma_X]] = \frac{|\Phi[\sigma_X]|_Y}{2^m}$ . In this case, it is immediate to see that solving (5) as a *DQMax#SAT* problem (i.e., by ignoring probabilities) would solve the original *DSSAT* problem. Otherwise, in the general case, one can use existing techniques such as [4] to transform arbitrary *DSSAT* problems (5) into equivalent ones where all probabilities are  $\frac{1}{2}$  and solve them as above.

Note that while the reduction above from *DSSAT* to *DQMax#SAT* seems to indicate the two problems are rather similar, a reverse reduction from *DQMax#SAT* to *DSSAT* seems not possible in general. That is, recall that *DQMax#SAT* allows for a third category of *existential* variables  $Z$  which can occur in the dependencies sets  $H_i$  and which are not used for counting but are projected out. Yet, such problems arise naturally in our application domain as illustrated later in section 6. If no such existential variables exists or if they do not occur in the dependencies sets then one can apriori project them from the objective  $\Phi$  and syntactically reduce *DQMax#SAT* to *DSSAT* (i.e., adding  $\frac{1}{2}$  probabilities on counting variables). However, projecting existential variables in a brute-force way may lead to an exponential blow-up of the objective formula  $\Phi$ , an issue already explaining the hardness of projected model counting vs model counting [2,17]. Otherwise, in case of dependencies on existential variables, it is an open question if any direct reduction exists as these variables do not fit into the two categories of variables (counting, maximizing) occurring in *DSSAT* formula.

### 3 Global method

We show in this section that the *DQMax#SAT* problem can be directly reduced to a *Max#SAT* problem with an exponentially larger number of maximizing variables and exponentially bigger objective formula.

First, recall that any boolean formula  $\varphi \in \mathcal{F}\langle H \rangle$  can be written as a finite disjunction of a subset  $M_\varphi$  of complete monomials from  $\mathcal{M}\langle H \rangle$ , that is, such that the following equivalences hold:

$$\varphi \iff \bigvee_{m \in M_\varphi} m \iff \bigvee_{m \in \mathcal{M}\langle H \rangle} (\llbracket m \in M_\varphi \rrbracket \wedge m)$$

Therefore, any formula  $\varphi \in \mathcal{F}\langle H \rangle$  is uniquely *encoded* by the set of boolean values  $\llbracket m \in M_\varphi \rrbracket$  denoting the membership of each complete monomial  $m$  to  $M_\varphi$ . We use this idea to encode the substitution of a maximizing variable  $x_i$  by some formula  $\varphi_i \in \mathcal{F}\langle H_i \rangle$  by using a set of boolean variables  $(x'_{i,m})_{m \in \mathcal{M}\langle H_i \rangle}$  denoting respectively  $\llbracket m \in M_{\varphi_i} \rrbracket$  for all  $m \in \mathcal{M}\langle H_i \rangle$ . We now define the following *Max#SAT* problem:

$$\begin{aligned} & (\max x'_{1,m \cdot})_{m \in \mathcal{M}\langle H_1 \rangle} \dots (\max x'_{n,m \cdot})_{m \in \mathcal{M}\langle H_n \rangle} \mathbf{R}Y. \exists Z. \exists X. \\ & \Phi(X, Y, Z) \wedge \bigwedge_{i \in [1, n]} (x_i \iff \bigvee_{m \in \mathcal{M}\langle H_i \rangle} (x'_{i,m} \wedge m)) \quad (6) \end{aligned}$$

The next theorem establishes the relation between the two problems.

**Theorem 1.**  $\sigma_X^* = \{x_i \mapsto \varphi_i^*\}_{i \in [1, n]}$  is a solution to the problem  $DQMax\#SAT$  (1) if and only if  $\alpha_{X'}^* = \{x'_{i, m} \mapsto \llbracket m \in M_{\varphi_i^*} \rrbracket\}_{i \in [1, n], m \in \mathcal{M}\langle H_i \rangle}$  is a solution to  $Max\#SAT$  problem (6).

*Proof.* Let us denote

$$\Phi'(X', X, Y, Z) \stackrel{def}{=} \Phi(X, Y, Z) \wedge \bigwedge_{i \in [1, n]} (x_i \Leftrightarrow \bigvee_{m \in \mathcal{M}\langle H_i \rangle} (x'_{i, m} \wedge m))$$

Actually, for any  $\Phi \in \mathcal{F}\langle X \cup Y \cup Z \rangle$  for any  $\varphi_1 \in \mathcal{F}\langle H_1 \rangle, \dots, \varphi_n \in \mathcal{F}\langle H_n \rangle$  the following equivalence is valid:

$$\begin{aligned} \Phi(X, Y, Z)[\{x_i \mapsto \varphi_i\}_{i \in [1, n]}] \Leftrightarrow \\ (\exists X. \Phi'(X', X, Y, Z)) [\{x'_{i, m} \mapsto \llbracket m \in M_{\varphi_i} \rrbracket\}_{i \in [1, n], m \in \mathcal{M}\langle H_i \rangle}] \end{aligned}$$

Consequently, finding the substitution  $\sigma_X$  which maximize the number of  $Y$ -models of the left-hand side formula (that is, of  $\exists Z. \Phi(X, Y, Z)$ ) is actually the same as finding the valuation  $\alpha_{X'}$  which maximizes the number of  $Y$ -models of the right-hand side formula (that is,  $\exists Z. \exists X. \Phi'(X', X, Y, Z)$ ).  $\square$

*Example 3.* Example 1 is reduced to the following:

$$\begin{aligned} \max x'_{1, z_1 z_2} \cdot \max x'_{1, z_1 \overline{z_2}} \cdot \max x'_{1, \overline{z_1} z_2} \cdot \max x'_{1, \overline{z_1} \overline{z_2}} \cdot \mathcal{R}y_1 \cdot \mathcal{R}y_2 \cdot \exists z_1 \cdot \exists z_2 \cdot \exists x_1 \cdot \\ (x_1 \Leftrightarrow y_1) \wedge (z_1 \Leftrightarrow y_1 \vee y_2) \wedge (z_2 \Leftrightarrow y_1 \wedge y_2) \wedge \\ (x_1 \Leftrightarrow ((x'_{1, z_1 z_2} \wedge z_1 \wedge z_2) \vee (x'_{1, z_1 \overline{z_2}} \wedge z_1 \wedge \overline{z_2}) \vee (x'_{1, \overline{z_1} z_2} \wedge \overline{z_1} \wedge z_2) \vee (x'_{1, \overline{z_1} \overline{z_2}} \wedge \overline{z_1} \wedge \overline{z_2}))) \end{aligned}$$

One possible answer is  $x'_{1, z_1 z_2} \mapsto \top, x'_{1, z_1 \overline{z_2}} \mapsto \top, x'_{1, \overline{z_1} z_2} \mapsto \perp, x'_{1, \overline{z_1} \overline{z_2}} \mapsto \perp$ . This yields the solution  $\sigma_X(x_1) = (z_1 \wedge z_2) \vee (z_1 \wedge \overline{z_2}) = z_1$  which is one of the optimal solutions as explained in Example 1.

## 4 Incremental method

In this section we propose a first improvement with respect to the reduction in the previous section. It allows to control the blow-up of the objective formula in the reduced  $Max\#SAT$  problem through an incremental process. Moreover, it allows in practice to find earlier good approximate solutions.

The incremental method consists in solving a sequence of related  $Max\#SAT$  problems, each one obtained from the original  $DQMax\#SAT$  problem and a reduced set of dependencies  $H'_1 \subseteq H_1, \dots, H'_n \subseteq H_n$ . Actually, if the sets of dependencies  $H'_1, \dots, H'_n$  are chosen such that to augment progressively from  $\emptyset, \dots, \emptyset$  to  $H_1, \dots, H_n$  by increasing only one of  $H'_i$  at every step then (i) it is possible to build every such  $Max\#SAT$  problem from the previous one by a simple syntactic transformation and (ii) most importantly, it is possible to steer the search for its solution knowing the solution of the previous one.

The incremental method relies therefore on an oracle procedure **max#sat** for solving *Max#SAT* problems. We assume this procedure takes as inputs the sets  $X, Y, Z$  of maximizing, counting and existential variables, an objective formula  $\Phi \in \mathcal{F}(X \cup Y \cup Z)$ , an initial assignment  $\alpha_0 : X \rightarrow \mathbb{B}$  and a filter formula  $\Psi \in \mathcal{F}(X)$ . The last two parameters are essentially used to restrict the search for maximizing solutions and must satisfy:

- $\Psi[\alpha_0] = \top$ , that is, the initial assignment  $\alpha_0$  is a model of  $\Psi$  and
- for all  $\alpha : X \rightarrow \mathbb{B}$  if  $\alpha \not\models \Psi$  then  $|\exists Z. \Phi[\alpha]|_Y \leq |\exists Z. \Phi[\alpha_0]|_Y$ , that is, any assignment  $\alpha$  outside the filter  $\Psi$  is at most as good as the assignment  $\alpha_0$ .

Actually, whenever the conditions hold, the oracle can safely restrict the search for the optimal assignments within the models of  $\Psi$ . The oracle produces as output the optimal assignment  $\alpha^* : X \rightarrow \mathbb{B}$  solving the *Max#SAT* problem.

The incremental algorithm proposed in Algorithm 1 proceeds as follows:

- at lines 1-5 it prepares the arguments for the first call of the *Max#SAT* oracle, that is, for solving the problem where  $H'_1 = H'_2 = \dots = H'_n = \emptyset$ ,
- at line 7 it calls to the *Max#SAT* oracle,
- at lines 9-10 it chooses an index  $i_0$  of some dependency set  $H'_i \neq H_i$  and a variable  $u \in H_{i_0} \setminus H'_{i_0}$  to be considered in addition for the next step,
- at lines 11-19 it prepares the argument for the next call of the *Max#SAT* oracle, that is, it updates the set of maximizing variables  $X'$ , it refines the objective formula  $\Phi'$ , it defines the new initial assignment  $\alpha'_0$  and the new filter  $\Psi'$  using the solution of the previous problem,
- at lines 6,20,22 it controls the main iteration, that is, keep going as long as sets  $H'_i$  are different from  $H_i$ ,
- at line 23 it builds the expected solution, that is, convert the Boolean solution  $\alpha'^*$  of the final *Max#SAT* problem where  $H'_i = H_i$  for all  $i \in [1, n]$  to the corresponding substitution  $\sigma_X^*$ .

Finally, note that the application of substitution at line 15 can be done such that to preserve the CNF form of  $\Phi'$ . That is, the substitution proceeds clause by clause by using the following equivalences, for every formula  $\psi$ :

$$\begin{aligned}
 (\psi \vee x'_{i_0, m})[x'_{i_0, m} \mapsto (x'_{i_0, mu} \wedge u) \vee (x'_{i_0, m\bar{u}} \wedge \bar{u})] &\Leftrightarrow \\
 (\psi \vee x'_{i_0, mu} \vee x'_{i_0, m\bar{u}}) \wedge (\psi \vee x'_{i_0, mu} \vee \bar{u}) \wedge (\psi \vee x'_{i_0, m\bar{u}} \vee u) & \\
 (\psi \vee \overline{x'_{i_0, m}})[x'_{i_0, m} \mapsto (x'_{i_0, mu} \wedge u) \vee (x'_{i_0, m\bar{u}} \wedge \bar{u})] &\Leftrightarrow (\psi \vee \overline{x'_{i_0, mu} \vee \bar{u}})(\psi \vee \overline{x'_{i_0, m\bar{u}} \vee u})
 \end{aligned}$$

**Theorem 2.** *Algorithm 1 is correct for solving the DQMax#SAT problem (1).*

*Proof.* The algorithm terminates after  $1 + \sum_{i \in [1, n]} |H_i|$  oracle calls. Moreover, every oracle call solves correctly the *Max#SAT* problem corresponding to DQ-*Max#SAT* problem

$$\max^{H'_1} x_1. \dots \max^{H'_n} x_n. \exists Y. \exists Z. \Phi(X, Y, Z)$$



```

input :  $X = \{x_1, \dots, x_n\}, Y, Z, H_1, \dots, H_n, \Phi$ 
output:  $\sigma_X^*$ 
1  $H'_i \leftarrow \emptyset$  for all  $i \in [1, n]$ 
2  $X' \leftarrow \{x'_{i,\top}\}_{i \in [1, n]}$ 
3  $\Phi' \leftarrow \Phi \wedge \bigwedge_{i \in [1, n]} (x_i \Leftrightarrow x'_{i,\top})$ 
4  $\alpha'_0 \leftarrow \{x'_{i,\top} \mapsto \perp\}_{i \in [1, n]}$ 
5  $\Psi' \leftarrow \top$ 
6 repeat
7    $\alpha'^* \leftarrow \text{max\#sat}(X', Y, Z \cup X, \Phi', \alpha'_0, \Psi')$ 
8   if  $H'_i \neq H_i$  for some  $i \in [1, n]$  then
9      $i_0 \leftarrow \text{choose}(\{i \in [1, n] \mid H'_i \neq H_i\})$ 
10     $u \leftarrow \text{choose}(H_{i_0} \setminus H'_{i_0})$ 
11     $\alpha'_0 \leftarrow \alpha'^*$ 
12     $\Psi' \leftarrow \perp$ 
13    foreach  $m \in \mathcal{M}\langle H'_{i_0} \rangle$  do
14       $X' \leftarrow (X' \setminus \{x'_{i_0, m}\}) \cup \{x'_{i_0, mu}, x'_{i_0, m\bar{u}}\}$ 
15       $\Phi' \leftarrow \Phi' [x'_{i_0, m} \mapsto (x'_{i_0, mu} \wedge u) \vee (x'_{i_0, m\bar{u}} \wedge \bar{u})]$ 
16       $\alpha'_0 \leftarrow (\alpha'_0 \setminus \{x'_{i_0, m} \mapsto \_ \}) \cup \{x'_{i_0, mu}, x'_{i_0, m\bar{u}} \mapsto \alpha'_0(x'_{i_0, m})\}$ 
17       $\Psi' \leftarrow \Psi' \vee (x'_{i_0, mu} \not\equiv x'_{i_0, m\bar{u}})$ 
18    end
19     $\Psi' \leftarrow \Psi' \vee \bigwedge_{x \in X'} (x \Leftrightarrow \alpha'_0(x))$ 
20     $H'_{i_0} \leftarrow H'_{i_0} \cup \{u\}$ 
21  end
22 until  $H'_i = H_i$  for all  $i \in [1, n]$ 
23  $\sigma_X^* \leftarrow \{x_i \mapsto \bigvee_{m \in \mathcal{M}\langle H_i \rangle} (\alpha'^*(x'_{i, m}) \wedge m)\}_{i \in [1, n]}$ 

```

**Algorithm 1:** Incremental Algorithm

This is an invariance property provable by induction. It holds by construction of  $X'$ ,  $\Phi'$ ,  $\alpha'_0$ ,  $\Psi'$  at the initial step. Then, it is preserved from one oracle call to the next one i.e.,  $X'$  and  $\Phi'$  are changed such that to reflect the addition of the variable  $u$  of the set  $H'_{i_0}$ . The new initial assignment  $\alpha'_0$  is obtained (i) by replicating the optimal value  $\alpha'^*(x'_{i_0, m})$  to the newly introduced  $x'_{i_0, mu}, x'_{i_0, m\bar{u}}$  variables derived from  $x'_{i_0, m}$  variable (line 16) and (ii) by keeping the optimal value  $\alpha'^*(x'_{i, m})$  for other variables (line 11). As such, for the new problem, the assignment  $\alpha'_0$  has exactly the same number of  $Y$ -projected models as the optimal assignment  $\alpha'^*$  had on the previous problem. The filter  $\Psi'$  is built such that to contain this new initial assignment  $\alpha'_0$  (line 19) as well as any other assignment that satisfies  $x'_{i_0, mu} \not\equiv x'_{i_0, m\bar{u}}$  for some monomial  $m$  (lines 12, 17). This construction guarantees that, any assignment which does not satisfy the filter  $\Psi'$  reduces precisely to an assignment of the previous problem, other than the optimal one  $\alpha'^*$ , and henceforth at most as good as  $\alpha'_0$  regarding the number of  $Y$ -projected models. Therefore, it is a sound filter and can be used to restrict the search for the new problem. The final oracle call corresponds to solving the complete *Max#SAT* problem (6) and it will therefore allow to derive a correct solution to the initial *DQMax#SAT* problem (1).  $\square$

*Example 4.* Let reconsider Example 1. The incremental algorithm will perform 3 calls to the *Max#SAT* oracle. The first call corresponds to the problem

$$\max x'_{1,\top}. \mathfrak{A}y_1. \mathfrak{A}y_2. \exists z_1. \exists z_2. \exists x_1. \\ (x_1 \Leftrightarrow y_1) \wedge (z_1 \Leftrightarrow y_1 \vee y_2) \wedge (z_2 \Leftrightarrow y_1 \wedge y_2) \wedge (x_1 \Leftrightarrow x'_{1,\top})$$

A solution found by the oracle is e.g.,  $x'_{1,\top} \mapsto \perp$  which has 2 projected models. If  $z_1$  is added to  $H'_1$ , the second call corresponds to the refined *Max#SAT* problem:

$$\max x'_{1,z_1}. \max x'_{1,\bar{z}_1}. \mathfrak{A}y_1. \mathfrak{A}y_2. \exists z_1. \exists z_2. \exists x_1. \\ (x_1 \Leftrightarrow y_1) \wedge (z_1 \Leftrightarrow y_1 \vee y_2) \wedge (z_2 \Leftrightarrow y_1 \wedge y_2) \wedge (x_1 \Leftrightarrow x'_{1,z_1} \wedge z_1 \vee x'_{1,\bar{z}_1} \wedge \bar{z}_1)$$

A solution found by the oracle is e.g.,  $x'_{1,z_1} \mapsto \top, x'_{1,\bar{z}_1} \mapsto \perp$  which has 3 projected models. Finally,  $z_2$  is added to  $H'_1$  therefore the third call corresponds to the complete *Max#SAT* problem as presented in Example 3. The solution found by the oracle is the same as in Example 3.

A first benefit of Algorithm 1 is the fact that it opens the door to any-time approaches to solve the *DQMax#SAT* problem. Indeed, the distance between the current and the optimal solution (that is, the relative ratio between the corresponding number of  $Y$ -projected models) can be estimated using the upper bound provided by Prop. 1. Hence, one could stop the search at any given iteration as soon as some threshold is reached and construct the returned value  $\sigma_X$  similarly as in Line 23 of Algorithm 1. In this case the returned  $\sigma_X$  would be defined as  $\sigma_X = \{x_i \mapsto \vee_{m \in \mathcal{M}(H'_i)} (\alpha'^*(x'_{i,m}) \wedge m)\}_{i \in [1,n]}$  (note here that the monomials are selected from  $H'_i$  instead of  $H_i$ ).

Another benefit of the incremental approach is that it is applicable without any assumptions on the underlying *Max#SAT* solver. Indeed, one can use  $\Psi'$  in Algorithm 1 by solving the *Max#SAT* problem corresponding to  $\Phi' \wedge \Psi'$ , and return the found solution. Even though the  $\alpha'_0$  parameter requires an adaptation of the *Max#SAT* solver in order to ease the search of a solution, one could still benefit from the incremental resolution of *DQMax#SAT*. Notice that a special handling of the  $\Psi'$  parameter by the solver would avoid complexifying the formula passed to the *Max#SAT* solver and still steer the search properly.

## 5 Local method

The local resolution method allows to compute the solution of an initial *DQMax#SAT* problem by combining the solutions of two strictly smaller and independent *DQMax#SAT* sub-problems derived syntactically from the initial one. The local method applies only if either 1) some counting or existential variable  $u$  is occurring in all dependency set; or 2) if there is some maximizing variable having an empty dependency set. That is, in contrast to the global and incremental methods, the local method is applicable only in specific situations.

Let us consider a *DQMax#SAT* problem of form (1). Given a variable  $v$ , let  $\Phi_v \stackrel{def}{=} \Phi[v \mapsto \top]$ ,  $\Phi_{\bar{v}} \stackrel{def}{=} \Phi[v \mapsto \perp]$  be the two cofactors on variable  $v$  of the objective  $\Phi$ .

### 5.1 Reducing common dependencies

Let us consider now a variable  $u$  which occurs in all dependency sets  $H_i$  and let us consider the following  $u$ -reduced  $DQMax\#SAT$  problems:

$$\max^{H_1 \setminus \{u\}} x_1. \dots \max^{H_n \setminus \{u\}} x_n. \mathfrak{R} Y \setminus \{u\}. \exists Z \setminus \{u\}. \Phi_u \quad (7)$$

$$\max^{H_1 \setminus \{u\}} x_1. \dots \max^{H_n \setminus \{u\}} x_n. \mathfrak{R} Y \setminus \{u\}. \exists Z \setminus \{u\}. \Phi_{\bar{u}} \quad (8)$$

Let  $\sigma_{X,u}^*$ ,  $\sigma_{X,\bar{u}}^*$  denote respectively the solutions to the problems above.

**Theorem 3.** *If either*

- (i)  $u \in Y$  or
- (ii)  $u \in Z$  and  $u$  is functionally dependent on counting variables  $Y$  within the objective  $\Phi$  (that is, for any valuation  $\alpha_Y : Y \rightarrow \mathbb{B}$ , at most one of  $\Phi[\alpha_Y][u \mapsto \top]$  and  $\Phi[\alpha_Y][u \mapsto \perp]$  is satisfiable).

then  $\sigma_X^*$  defined as

$$\sigma_X^*(x_i) \stackrel{def}{=} (u \wedge \sigma_{X,u}^*(x_i)) \vee (\bar{u} \wedge \sigma_{X,\bar{u}}^*(x_i)) \text{ for all } i \in [1, n]$$

is a solution to the  $DQMax\#SAT$  problem (1).

*Proof.* First, any formula  $\varphi_i \in \mathcal{F}\langle H_i \rangle$  can be equivalently written as  $u \wedge \varphi_{i,u} \vee \bar{u} \wedge \varphi_{i,\bar{u}}$  where  $\varphi_{i,u} \stackrel{def}{=} \varphi_i[u \mapsto \top] \in \mathcal{F}\langle H_i \setminus \{u\} \rangle$  and  $\varphi_{i,\bar{u}} \stackrel{def}{=} \varphi_i[u \mapsto \perp] \in \mathcal{F}\langle H_i \setminus \{u\} \rangle$ . Second, we can prove the equivalence:

$$\begin{aligned} \Phi[x_i \mapsto \varphi_i] &\Leftrightarrow (u \wedge \Phi_u \vee \bar{u} \wedge \Phi_{\bar{u}})[x_i \mapsto u \wedge \varphi_{i,u} \vee \bar{u} \wedge \varphi_{i,\bar{u}}] \\ &\Leftrightarrow u \wedge \Phi_u[x_i \mapsto \varphi_{i,u}] \vee \bar{u} \wedge \Phi_{\bar{u}}[x_i \mapsto \varphi_{i,\bar{u}}] \end{aligned}$$

by considering the decomposition of  $\Phi_u$ ,  $\Phi_{\bar{u}}$  according to the variable  $x_i$ . The equivalence above can then be generalized to a complete substitution  $\sigma_X = \{x_i \mapsto \varphi_i\}_{i \in [1, n]}$  of maximizing variables. Let us denote respectively  $\sigma_{X,u} \stackrel{def}{=} \{x_i \mapsto \varphi_{i,u}\}_{i \in [1, n]}$ ,  $\sigma_{X,\bar{u}} \stackrel{def}{=} \{x_i \mapsto \varphi_{i,\bar{u}}\}_{i \in [1, n]}$ . Therefore, one obtains

$$\begin{aligned} \Phi[\sigma_X] &\Leftrightarrow (u \wedge \Phi_u \vee \bar{u} \wedge \Phi_{\bar{u}})[x_i \mapsto \varphi_i]_{i \in [1, n]} \\ &\Leftrightarrow u \wedge \Phi_u[\sigma_{X,u}] \vee \bar{u} \wedge \Phi_{\bar{u}}[\sigma_{X,\bar{u}}] \\ &\Leftrightarrow u \wedge \Phi_u[\sigma_{X,u}] \vee \bar{u} \wedge \Phi_{\bar{u}}[\sigma_{X,\bar{u}}] \end{aligned}$$

Third, the later equivalence provides a way to compute the number of  $Y$ -models of the formula  $\exists Z. \Phi[\sigma_Z]$  as follows:

$$\begin{aligned} |\exists Z. \Phi[\sigma_Z]|_Y &= |\exists Z. (u \wedge \Phi_u[\sigma_{X,u}] \vee \bar{u} \wedge \Phi_{\bar{u}}[\sigma_{X,\bar{u}}])|_Y \\ &= |\exists Z. (u \wedge \Phi_u[\sigma_{X,u}]) \vee \exists Z. (\bar{u} \wedge \Phi_{\bar{u}}[\sigma_{X,\bar{u}}])|_Y \\ &= |\exists Z. (u \wedge \Phi_u[\sigma_{X,u}])|_Y + |\exists Z. (\bar{u} \wedge \Phi_{\bar{u}}[\sigma_{X,\bar{u}}])|_Y \\ &= |\exists Z \setminus \{u\}. \Phi_u[\sigma_{X,u}]|_{Y \setminus \{u\}} + |\exists Z \setminus \{u\}. \Phi_{\bar{u}}[\sigma_{X,\bar{u}}]|_{Y \setminus \{u\}} \end{aligned}$$

Note that the third equality holds only because  $u \in Y$  or  $u \in Z$  and functionally dependent on counting variables  $Y$ . Actually, in these situations, the sets of  $Y$ -projected models of respectively,  $u \wedge \Phi_u[\sigma_{X,u}]$  and  $\bar{u} \wedge \Phi_{\bar{u}}[\sigma_{X,\bar{u}}]$  are disjoint. Finally, the last equality provides the justification of the theorem, that is, finding  $\sigma_X$  which maximizes the left hand side reduces to finding  $\sigma_{X,u}$ ,  $\sigma_{X,\bar{u}}$  which maximizes independently the two terms of right hand side, and these actually are the solutions of the two  $u$ -reduced problems (7) and (8).  $\square$

*Example 5.* Let us reconsider Example 1. It is an immediate observation that existential variables  $z_1, z_2$  are functionally dependent on counting variables  $y_1, y_2$  according to the objective. Therefore the local method is applicable and henceforth since  $H_1 = \{z_1, z_2\}$  one reduces the initial problem to four smaller problems, one for each valuation of  $z_1, z_2$ , as follows:

$$\begin{aligned} z_1 \mapsto \top, z_2 \mapsto \top : & \max^0 x_1. \mathcal{R}y_1. \mathcal{R}y_2. (x_1 \Leftrightarrow y_1) \wedge (\top \Leftrightarrow y_1 \vee y_2) \wedge (\top \Leftrightarrow y_1 \wedge y_2) \\ z_1 \mapsto \top, z_2 \mapsto \perp : & \max^0 x_1. \mathcal{R}y_1. \mathcal{R}y_2. (x_1 \Leftrightarrow y_1) \wedge (\top \Leftrightarrow y_1 \vee y_2) \wedge (\perp \Leftrightarrow y_1 \wedge y_2) \\ z_1 \mapsto \perp, z_2 \mapsto \top : & \max^0 x_1. \mathcal{R}y_1. \mathcal{R}y_2. (x_1 \Leftrightarrow y_1) \wedge (\perp \Leftrightarrow y_1 \vee y_2) \wedge (\top \Leftrightarrow y_1 \wedge y_2) \\ z_2 \mapsto \perp, z_2 \mapsto \perp : & \max^0 x_1. \mathcal{R}y_1. \mathcal{R}y_2. (x_1 \Leftrightarrow y_1) \wedge (\perp \Leftrightarrow y_1 \vee y_2) \wedge (\perp \Leftrightarrow y_1 \wedge y_2) \end{aligned}$$

The four problems are solved independently and have solutions e.g., respectively  $x_1 \mapsto c_1 \in \{\top\}$ ,  $x_1 \mapsto c_2 \in \{\top, \perp\}$ ,  $x_1 \mapsto c_3 \in \{\top, \perp\}$ ,  $x_1 \mapsto c_4 \in \{\perp\}$ . By recombining these solutions according to Theorem 3 one obtains several solutions to the original  $DQMax\#SAT$  problem of the form:

$$x_1 \mapsto (z_1 \wedge z_2 \wedge c_1) \vee (z_1 \wedge \bar{z}_2 \wedge c_2) \vee (\bar{z}_1 \wedge z_2 \wedge c_3) \vee (\bar{z}_1 \wedge \bar{z}_2 \wedge c_4)$$

They correspond to solutions already presented in Example 3, that is:

$$\begin{aligned} x_1 \mapsto (z_1 \wedge z_2 \wedge \top) \vee (z_1 \wedge \bar{z}_2 \wedge \perp) \vee (\bar{z}_1 \wedge z_2 \wedge \perp) \vee (\bar{z}_1 \wedge \bar{z}_2 \wedge \perp) & (\equiv z_1 \wedge z_2) \\ x_1 \mapsto (z_1 \wedge z_2 \wedge \top) \vee (z_1 \wedge \bar{z}_2 \wedge \perp) \vee (\bar{z}_1 \wedge z_2 \wedge \top) \vee (\bar{z}_1 \wedge \bar{z}_2 \wedge \perp) & (\equiv z_2) \\ x_1 \mapsto (z_1 \wedge z_2 \wedge \top) \vee (z_1 \wedge \bar{z}_2 \wedge \top) \vee (\bar{z}_1 \wedge z_2 \wedge \perp) \vee (\bar{z}_1 \wedge \bar{z}_2 \wedge \perp) & (\equiv z_1) \\ x_1 \mapsto (z_1 \wedge z_2 \wedge \top) \vee (z_1 \wedge \bar{z}_2 \wedge \top) \vee (\bar{z}_1 \wedge z_2 \wedge \top) \vee (\bar{z}_1 \wedge \bar{z}_2 \wedge \perp) & (\equiv z_1 \vee z_2) \end{aligned}$$

Finally, note that the local resolution method has potential for parallelization. It is possible to eliminate not only one but all common variables in the dependency sets as long as they fulfill the required property. This leads to several strictly smaller sub-problems that can be solved in parallel. The situation has been already illustrated in the previous example, where by the elimination of  $z_1$  and  $z_2$  one obtains 4 smaller sub-problems.

## 5.2 Solving variables with no dependencies

Let us consider now a maximizing variable which has an empty dependency set. Without lack of generality, assume  $x_1$  has an empty dependency set, i.e.  $H_1 = \emptyset$ .

Thus, the only possible values that can be assigned to  $x_1$  are  $\top$  or  $\perp$ . Let us consider the following  $x_1$ -reduced  $DQMax\#SAT$  problems:

$$\begin{aligned} \max^{H_2} x_2. \dots \max^{H_n} x_n. \Re Y. \exists Z. \Phi_{x_1} \\ \max^{H_2} x_2. \dots \max^{H_n} x_n. \Re Y. \exists Z. \Phi_{\overline{x_1}} \end{aligned}$$

and let  $\sigma_{X,x_1}^*, \sigma_{X,\overline{x_1}}^*$  denote respectively the solutions to the problems above. The following proposition is easy to prove, and provides the solution of the original problem based on the solutions of the two smaller sub-problems.

**Proposition 2.** *The substitution  $\sigma_X^*$  defined as*

$$\sigma_X^* \stackrel{def}{=} \begin{cases} \sigma_{X,x_1}^* \uplus \{x_1 \mapsto \top\} & \text{if } |\exists Z. \Phi_{x_1}[\sigma_{X,x_1}^*]|_Y \geq |\exists Z. \Phi_{\overline{x_1}}[\sigma_{X,\overline{x_1}}^*]|_Y \\ \sigma_{X,\overline{x_1}}^* \uplus \{x_1 \mapsto \perp\} & \text{otherwise} \end{cases}$$

*is a solution to the  $DQMax\#SAT$  problem (1).*

## 6 Application to Software Security

In this section, we give a concrete application of  $DQMax\#SAT$  in the context of *software security*. More precisely, we show that finding an optimal strategy for an adaptative attacker trying to break the security of some program can be naturally encoded as specific instances of the  $DQMax\#SAT$  problem.

In our setting, we allow the attacker to interact multiple times with the target program. Moreover, we assume that the adversary is able to make *observations*, either from the legal outputs or using some side-channel leaks. Adaptive attackers [9,24,23] are a special form of active attackers considered in security that are able to select their inputs based on former observations, such that they maximize their chances to reach their goals (i.e., break some security properties).

First we present in more details this attacker model we consider, and then we focus on two representative attack objectives the attacker aims to maximize:

- either the probability of reaching a specific point in the target program, while satisfying some objective function (Section 6.2),
- or the amount of information it can get about some fixed secret used by the program (Section 6.3).

At the end of the section, we show that the improvements presented in the previous sections apply in both cases.

### 6.1 Our model of security in presence of an adaptive adversary

The general setting we consider is the one of so-called *active* attackers, able to provide *inputs* to the program they target. Such attacks are then said *adaptive* when the attacker is able to deploy an attack strategy, which continuously relies on some knowledge gained from previous interactions with the target program,

and allowing to maximize its chances of success. Moreover, we consider the more powerful attacker model where the adversary is assumed to know the code of the target program.

Note that such an attacker model is involved in most recent concrete attack scenarios, where launching an exploit or disclosing some sensitive data requires to chain several (interactive) attack steps in order to defeat some protections and/or to gain some intermediate privileges on the target platform. Obviously, from the defender side, quantitative measures about the “controllability” of such attacks is of paramount importance for exploit analysis or vulnerability triage.

When formalizing the process of *adaptatively attacking* a given program, one splits the program’s variables between those *controlled* and those *uncontrolled* by the attacker. Among the *uncontrolled* variables one further distinguishes those *observable* and those *non-observable*, the former ones being available to the attacker for producing its (next) inputs. The *objective* of the attacker is a formula, depending on the values of program variables, and determining whether the attacker has successfully conducted the attack.

For the sake of simplicity – in our examples – we restrict ourselves to non-looping sequential programs operating on variables with bounded domains (such as finite integers, Boolean’s, etc). We furthermore consider the programs are written in SSA form, assuming that each variable is assigned before it is used. These hypothesis fit well in the context of a code analysis technique like *symbolic execution* [15], extensively used in software security.

Finally, we also rely on explicit (user-given) annotations by predefined functions (or macros) to identify the different classes of program variables and the attacker’s objective. In the following code excerpts, we assume that:

- The **random** function produces an uncontrolled non-observable value; it allows for instance to simulate the generation of both long term keys and nonces in a program using cryptographic primitives.
- The **input** function feeds the program with an attacker-controlled value.
- The **output** function simulates an observation made by the adversary and denotes a value obtained through the evaluation of some expression of program variables.

## 6.2 Security as a reachability property

We show in this section how to encode *quantitative reachability* defined in [3] as an instance of the *DQMax#SAT* problem.

In *quantitative reachability*, the goal of an adversary is to reach some target location in some program such that some *objective property* get satisfied. In order to model this target location of the program that the attacker wants to reach, we extend our simple programming language with a distinguished **win** function. The **win** function can take a predicate as argument (the objective property) and is omitted whenever this predicate is the **True** predicate. In practice such a predicate may encode some extra conditions required to trigger and exploit some vulnerability at the given program location (e.g., overflowing a buffer with a given payload).

```

1  $y_1 \leftarrow \text{random}()$ 
2  $y_2 \leftarrow \text{random}()$ 
3  $z_1 \leftarrow \text{output}(y_1 + y_2)$ 
4  $x_1 \leftarrow \text{input}()$ 
5 if  $y_1 \leq x_1$  then
6   |  $\text{win}(x_1 \leq y_2)$ 
7 end

```

**Program 2:** A first program example

*Example 6.* In Program 2 one can see an example of annotated program.  $y_1$  and  $y_2$  are uncontrollable non-observable variables.  $z_1$  is an observable variable holding the sum  $y_1 + y_2$ .  $x_1$  is a variable controlled by the attacker. The *attacker's objective* corresponds to the *path predicate*  $y_1 \leq x_1$  denoting the condition to reach the **win** function call and the *argument predicate*  $x_1 \leq y_2$  denoting the *objective property*. Let us observe that a successful attack exists, that is, by taking  $x_1 \leftarrow \frac{z_1}{2}$  the objective is always reachable.

When formalizing adaptive attackers, the *temporality* of interactions (that is, the order of inputs and outputs) is important, as the attacker can only synthesize an input value from the output values that were observed *before* it is asked to provide that input. To track the temporal dependencies in our formalization, for every controlled variable  $x_i$  one considers the set  $H_i$  of observable variables effectively known at the time of defining  $x_i$ , that is, representing the accumulation of attacker's knowledge throughout the interactions with the program.

We propose hereafter a systematic way to express the problem of synthesis of an optimal attack (that is, with the highest probability of the objective property to get satisfied), as a *DQMax#SAT* instance. Let  $Y$  (resp.  $Z$ ) be the set of uncontrolled variables being assigned to **random**() which in this section is assumed to uniformly sample values in their domain (resp. other expressions) in the program. For a variable  $z \in Z$  let moreover  $e_z$  be the unique expression assigned to it in the program, either through an assignment of the form  $z \leftarrow e_z$  or  $z \leftarrow \text{output}(e_z)$ . Let  $X = \{x_1, \dots, x_n\}$  be the set of controlled variables with their temporal dependencies respectively subsets  $H_1, \dots, H_n \subseteq Z$  of uncontrollable variables. Finally, let  $\Psi$  be the attacker objective, that is, the conjunction of the argument of the **win** function and the path predicate leading to the **win** function call. Consider the next most likely generalized *DQMax#SAT* problem:

$$\max^{H_1} x_1. \dots \max^{H_n} x_n. \Re Y. \exists Z. \Psi \wedge \bigwedge_{z \in Z} (z = e_z) \quad (9)$$

*Example 7.* Consider the annotated problem from Program 2. The encoding of the optimal attack leads to the generalized *DQMax#SAT* problem:

$$\max^{\{z_1\}} x_1. \Re y_1. \Re y_2. \exists z_1. (y_1 \leq x_1 \wedge x_1 \leq y_2) \wedge (z_1 = y_1 + y_2)$$

Note that in contrast to the *DQMax#SAT* problem (1), the variables are not restricted to Booleans (but to some finite domains) and the expressions

are not restricted to Boolean terms (but involve additional operators available in the specific domain theories e.g.,  $=$ ,  $\geq$ ,  $+$ ,  $-$ , etc). Nevertheless, as long as both variables and additional operators can be respectively, represented by and interpreted as operations on bitvectors, one can use *bitblasting* and transforms the generalized problem into a full-fledged *DQMax#SAT* problem and then solve it by the techniques introduced earlier in the paper.

Finally, note also that in the *DQMax#SAT* problems constructed as above, the maximizing variables are dependent by definition on existential variables only. Therefore, as earlier discussed in Section 2, these problems cannot be actually reduced to similar *DSSAT* problems. However, they compactly encode the quantitative reachability properties subject to input/output dependencies.

### 6.3 Security as a lack of leakage property

In this section, we extend earlier work on adaptive attackers from [24] by effectively synthesizing the *strategy* the attacker needs to deploy in order to maximize its knowledge about some secret value used by the program. Moreover, we show that in our case, we are able to keep symbolic the trace corresponding to the attack strategy, while in [23], the attacker strategy is a concretized tree, which explicitly states, for each concrete program output, what should be the next input provided by the adversary. Following ideas proposed in [23], symbolic execution can be used to generate constraints characterizing partitions on the secrets values, where each partition corresponds to the set of secrets leading to the same *sequences* of side-channel observations.

```

1  $z \leftarrow \text{random}()$  //the secret
2  $x \leftarrow \text{input}()$ 
3 if  $x \geq z$  then
4   | ... some computation taking
   |   10 seconds
5 else
6   | ... some computation taking
   |   20 seconds
7 end
```

**Program 3:** A leaking program

```

1  $z \leftarrow \text{random}()$  ;
2  $x_1 \leftarrow \text{input}()$ ;
3  $y_1 \leftarrow \text{output}(x_1 \geq z)$ ;
4  $x_2 \leftarrow \text{input}()$ ;
5  $y_2 \leftarrow \text{output}(x_2 \geq z)$ ;
6  $x_3 \leftarrow \text{input}()$ ;
7  $y_3 \leftarrow \text{output}(x_3 \geq z)$ ;
```

**Program 4:** An iterated leaking program

*Example 8.* Let us consider the excerpt Program 3 taken from [23]. This program is not *constant-time*, namely it executes a branching instruction whose condition depends on the secret  $z$ . Hence an adversary able to learn the branch taken during the execution, either by measuring the time or doing some cache-based attack, will get some information about the secret  $z$ . A goal of an adversary interacting several times with the program could be to maximize the amount of information leaked about the secret value  $z$ . When the program is seen as a channel leaking information, the channel capacity theorem [25] states that the information leaked by a program is upper-bounded by the number of different



observable outputs of the program (and the maximum is achieved whenever the secret is the unique randomness used by the program). In our case, it means that an optimal adaptive adversary interacting  $k$ -times with the program should maximize the number of different observable outputs. Hence, for example, if as in [23], we fix  $k = 3$  and if we assume that the secret  $z$  is uniformly sampled in the domain  $1 \leq z \leq 6$ , then the optimal strategy corresponds to maximize the number of different observable outputs  $y$  of the Program 4, which corresponds to the following *DQMax#SAT* instance:

$$\max^0 x_1. \max^{\{y_1\}} x_2. \max^{\{y_1, y_2\}} x_3. \mathcal{R}y_1. \mathcal{R}y_2. \mathcal{R}y_3. \exists z. \\ (y_1 \Leftrightarrow x_1 \geq z) \wedge (y_2 \Leftrightarrow x_2 \geq z) \wedge (y_3 \Leftrightarrow x_3 \geq z) \wedge (1 \leq z \leq 6)$$

Our prototype provided the following solution:  $x_1 = 100, x_2 = y_1 10, x_3 = y_1 y_2 1$ , that basically says: the attacker should first input 4 to the program, then the input corresponding to the integer whose binary encoding is  $y_1$  concatenated with 10, and the last input  $x_3$  is the input corresponding to the integer whose binary encoding is the concatenation of  $y_2, y_1$  and 1. In [23] the authors obtain an equivalent attack encoded as a tree-like strategy of concrete values.

We now show a systematic way to express the problem of the synthesis of an optimal attack expressed as the maximal channel capacity of a program seen as an information leakage channel, as a *DQMax#SAT* instance. Contrary to the previous section, the roles of  $Y$  and  $Z$  are now switched:  $Y$  is a set of variables encoding the observables output by the program;  $Z$  is the set of variables uniformly sampled by `random()` or assigned to other expressions in the program. For a variable  $y \in Y$ , let  $e_y$  be the unique expression assigned to it in the program through an assignment of the form  $y \leftarrow \text{output}(e_y)$ . For a variable  $z \in Z$ , let moreover  $e_z$  be the unique expression assigned to it in the program through an assignment of the form  $z \leftarrow e_z$  or the constraint encoding the domain used to sample values in  $z \leftarrow \text{random}()$ . Let  $X = \{x_1, \dots, x_n\}$  be the set of controlled variables with their temporal dependencies respectively subsets  $H_1, \dots, H_n \subseteq Y$ . Consider now the following most likely generalized *DQMax#SAT* problem:

$$\max^{H_1} x_1. \dots \max^{H_n} x_n. \mathcal{R}Y. \exists Z. \bigwedge_{y \in Y} (y = e_y) \wedge \bigwedge_{z \in Z} (z = e_z)$$

Finally, in contrast to reachability properties, in the *DQMax#SAT* problems obtained as above for evaluating leakage properties, the maximizing variables are by definition dependent on counting variables only. Consequently, for these problems, the existential variables can be apriori eliminated so that to obtain an equivalent *DSSAT*<sup>1</sup> problem as discussed in Section 2.

#### 6.4 Some remarks about the applications to security

Let us notice some interesting properties of the attacker synthesis's *DQMax#SAT* problems. If controlled variables  $x_1, x_2, \dots, x_n$  are input in this order within the

<sup>1</sup> Actually these problems can even be reduced to *SSAT* instances.

program then necessarily  $H_1 \subseteq H_2 \subseteq \dots \subseteq H_n$ . That is, the knowledge of the attacker only increases as long as newer observable values became available to it. Moreover, since we assumed that variables are used only after they were initialized, the sets  $H_i$  contain observable variables that are dependent only on the counting variables  $Y$ . Hence we can apply iteratively the following steps from the local resolution method described in Section 5:

- While  $H_1 \neq \emptyset$ , apply the local resolution method described in Section 5.1 iteratively until  $H_1$  becomes empty. For example, it is the case of Example 6 where  $z_1$  is dependent only on counting variables  $y_1$  and  $y_2$ .
- When  $H_1$  becomes  $\emptyset$ , apply the local resolution method described in Section 5.2 in order to eliminate the first maximizing variable.

## 7 Implementation and Experiments

We implement Algorithm 1 leaving generic the choice of the underlying *Max#SAT* solver. For concrete experiments, we used both the approximate solver BAXMC<sup>2</sup> [26] and the exact solver D4MAX [1].

In the implementation of Algorithm 1 in our tool, the filter  $\Psi'$  is handled as discussed at the end of Section 4: the formula effectively solved is  $\Phi' \wedge \neg\Psi'$ , allowing to use any *Max#SAT* solver without any prior modification. Remark that none of BAXMC and D4MAX originally supported exploiting the  $\alpha_0$  parameter of Algorithm 1 out of the box. While D4MAX is used of the shelf, we modified BAXMC to actually support this parameter for the purpose of the experiment.

We use the various examples used in this paper as benchmark instances for the implemented tool. Examples 1 and 2 are used as they are. We furthermore use Example 11 (in appendix) which is a slightly modified version of Example 1. We consider Examples 7 and 8 from Section 6 and perform the following steps to convert them into *DQMax#SAT* instances: (i) bitblast the formula representing the security problem into a *DQMax#SAT* instance over boolean variables; (ii) solve the later formula; (iii) propagate the synthesized function back into a function over bit-vectors for easier visual inspection of the result.

We also add the following security related problems (which respectively correspond to Program 5 in appendix and a relaxed version of Example 8 in Section 6) into our benchmark set:

$$\text{Example 9. } \max^0 x_1. \max^{\{z_1\}} x_2. \max^{\{z_1, z_2\}} x_3. \mathfrak{R}y_1. \exists z_1. \exists z_2. \\ (x_3 = y_1) \wedge (z_1 = x_1 \geq y_1 \wedge z_2 = x_2 \geq y)$$

$$\text{Example 10. } \max^0 x_1. \max^{\{y_1\}} x_2. \max^{\{y_1, y_2\}} x_3. \mathfrak{R}y_1. \mathfrak{R}y_2. \mathfrak{R}y_3. \exists z. \\ (y_1 \Leftrightarrow x_1 \geq z) \wedge (y_2 \Leftrightarrow x_2 \geq z) \wedge (y_3 \Leftrightarrow x_3 \geq z)$$

When bitblasting is needed for a given benchmark, the number of bits used for bitblasting is indicated in parentheses. After the bitblasting operation, the problems can be considered medium sized.

<sup>2</sup> Thanks to specific parametrization and the oracles [5] used internally by BAXMC, it can be considered an exact solver on the small instances of interest in this section.

**Table 1.** Summary of the performances of the tool.  $|\Phi|$  denotes the number of clauses. The last two columns indicate the running time using the specific *Max#SAT* oracle.

| Benchmark name      | $ X $ | $ Y $ | $ Z $ | $ \Phi $ | Time (BAXMC) | Time (D4MAX) |
|---------------------|-------|-------|-------|----------|--------------|--------------|
| Example 1           | 1     | 2     | 2     | 7        | 32ms         | 121ms        |
| Example 2           | 2     | 2     | 2     | 7        | 25ms         | 134ms        |
| Example 11          | 1     | 2     | 1     | 5        | 16ms         | 89ms         |
| Example 7 (3 bits)  | 3     | 6     | 97    | 329      | 378ms        | 79.88s       |
| Example 7 (4 bits)  | 4     | 8     | 108   | 385      | 638.63s      | > 30mins     |
| Example 8 (3 bits)  | 9     | 3     | 150   | 487      | 18.78s       | 74.58s       |
| Example 9 (3 bits)  | 9     | 3     | 93    | 289      | 74.00s       | 18.62s       |
| Example 10 (3 bits) | 9     | 3     | 114   | 355      | 9.16s        | 93.48s       |

As you can see in Table 1, the implemented tool can effectively solve all the examples presented in this paper. The synthesized answers (i.e. the monomials selected in Algorithm 1, Line 23) returned by both oracles are the same.

For security examples, one key part of the process is the translation of the synthesized answer (over boolean variables) back to the original problem (over bit-vectors). In order to do that, one can simply concatenate the generated sub-functions for each bit of the bit-vector into a complete formula, but that would lack explainability because the thus-generated function would be a concatenation of potentially big sums of monomials. In order to ease visual inspection, we run a generic simplification step [13] for all the synthesized sub-function, before concatenation. This simplification allows us to directly derive the answers explicated in Examples 7 and 8 instead of their equivalent formulated as sums of monomials, and better explain the results returned by the tool.

Unfortunately, we could not compare our algorithm against the state-of-the-art *DSSAT* solver *DSSATPRE* [18] on the set of example described in this paper because (i) as discussed in Section 2.3, some *DQMax#SAT* instances cannot be converted into *DSSAT* instances, (ii) for the only *DQMax#SAT* instance (Example 10) that can be converted into a *DSSAT* instance, we were not able to get an answer using *DSSATPRE*.

## 8 Related Work

As shown in Section 2, *DQMax#SAT* subsumes the *DSSAT* and *DQBF* problems. This relation indicates a similarity of the three problems, and thus some related works can be extracted from here. From the complexity point of view, the decision version of *DQMax#SAT* can be shown to be *NEXPTIME*-complete and hence it lies in the same complexity class as *DQBF* [21] and *DSSAT* [18].

Comparing the performances of existing *DQBF* algorithms with the proposed algorithms for *DQMax#SAT* is not yet realistic since they address different objectives. However, one can search for potential improvements for solving *DQMax#SAT* by considering the existing enhancements proposed in [16] to improve the resolution of *DQBF*. For example, dependency schemes [28] are a way

to change the dependency sets in *DQBF* without changing the *truth value* compared to the original formula. Thus, adaptations of these dependency schemes could be applied to our problem as well and potentially lead to a significant decrease of the size of the resulting *Max#SAT* problems.

The *DSSAT* problem is currently receiving an increased attention by the research community. A first sound and complete resolution procedure has been proposed in [19], however, without being yet implemented. The only available *DSSAT* solver nowadays is *DSSATPRE* [8]. This tool relies on preprocessing to get rid of dependencies and to produce equivalent *SSAT* problems. These problems are then accurately solved by existing *SSAT* solvers [7,27], some of them being also able to compute the optimal assignments for maximizing variables. In contrast, our tool for solving *DQMax#SAT* relies on existing *Max#SAT* solvers, always synthesizes the assignments for maximizing variables and provide support for approximate solving. Moreover, due to the presence of existential variables, note that *DQMax#SAT* and *DSSAT* are fundamentally different problems. Existential variables are already pinpointing the difference between the two pure counting problems *#SAT* and *# $\exists$ SAT* [2,17]. In cases where maximizing variables depend on existential variables no trivial reduction from *DQMax#SAT* to *DSSAT* seems to exists.

From the security point of view, the closest works to our proposal are the ones decribed in [23,24]. As the authors in these papers, we are able to effectively synthesize the optimal adaptive strategy the attacker needs to deploy in order to maximize its knowledge about some secret value used by the program. In addition, we show that in our case, we are able to keep symbolic the trace corresponding to the attack strategy, while in [23], the attacker strategy is a concretized tree which explicitly states, for each concrete program output, what should be the next input provided by the adversary.

## 9 Conclusions

We exposed in this paper a new problem called *DQMax#SAT* that subsumes both *DQBF* and *DSSAT*. We then devised three different resolution methods based on reductions to *Max#SAT* and showed the effectiveness of one of them, the incremental method, by implementing a prototype solver. A concrete application of *DQMax#SAT* lies in the context of software security, in order to assess the robustness of a program by synthesizing the optimal adversarial strategy of an adaptive attacker.

Our work can be expanded in several directions. First, we would like to enhance our prototype with strategies for dependency expansion in the incremental algorithm. Second, we plan to integrate the local resolution method in our prototype. Third, we shall apply these techniques on more realistic security related examples, and possibly getting further improvement directions from this dedicated context.

## References

1. Audemard, G., Lagniez, J., Miceli, M.: A New Exact Solver for (Weighted) Max#SAT. In: SAT. LIPIcs, vol. 236, pp. 28:1–28:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)
2. Aziz, R.A., Chu, G., Muise, C.J., Stuckey, P.J.: # $\exists$ SAT: Projected Model Counting. In: SAT. Lecture Notes in Computer Science, vol. 9340, pp. 121–137. Springer (2015)
3. Bardin, S., Girol, G.: A Quantitative Flavour of Robust Reachability. CoRR **abs/2212.05244** (2022)
4. Chakraborty, S., Fried, D., Meel, K.S., Vardi, M.Y.: From Weighted to Unweighted Model Counting. In: IJCAI. pp. 689–695. AAAI Press (2015)
5. Chakraborty, S., Meel, K.S., Vardi, M.Y.: A Scalable Approximate Model Counter. CoRR **abs/1306.5726** (2013)
6. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Balancing Scalability and Uniformity in SAT Witness Generator. In: DAC. pp. 60:1–60:6. ACM (2014)
7. Chen, P., Huang, Y., Jiang, J.R.: A Sharp Leap from Quantified Boolean Formula to Stochastic Boolean Satisfiability Solving. In: AAAI. pp. 3697–3706. AAAI Press (2021)
8. Cheng, C., Jiang, J.R.: Lifting (D)QBF Preprocessing and Solving Techniques to (D)SSAT. In: AAAI. pp. 3906–3914. AAAI Press (2023)
9. Dullien, T.: Weird Machines, Exploitability, and Provable Unexploitability. IEEE Trans. Emerg. Top. Comput. **8**(2), 391–403 (2020)
10. Fremont, D.J., Rabe, M.N., Seshia, S.A.: Maximum Model Counting. In: AAAI. pp. 3885–3892. AAAI Press (2017)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
12. Garey, M.R., Johnson, D.S., So, H.C.: An Application of Graph Coloring to Printed Circuit Testing (Working Paper). In: FOCS. pp. 178–183. IEEE Computer Society (1975)
13. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT workshop. vol. 2015 (2015)
14. Henkin, L., Karp, C.R.: Some Remarks on Infinitely Long Formulas. Journal of Symbolic Logic **30**(1), 96–97 (1965). <https://doi.org/10.2307/2270594>
15. King, J.C.: Symbolic Execution and Program Testing. Commun. ACM **19**(7), 385–394 (1976)
16. Kovásznai, G.: What is the state-of-the-art in DQBF solving. In: MaCS-16. Joint Conference on Mathematics and Computer Science (2016)
17. Lagniez, J., Marquis, P.: A Recursive Algorithm for Projected Model Counting. In: AAAI. pp. 1536–1543. AAAI Press (2019)
18. Lee, N., Jiang, J.R.: Dependency Stochastic Boolean Satisfiability: A Logical Formalism for NEXPTIME Decision Problems with Uncertainty. In: AAAI. pp. 3877–3885. AAAI Press (2021)
19. Luo, Y., Cheng, C., Jiang, J.R.: A Resolution Proof System for Dependency Stochastic Boolean Satisfiability. J. Autom. Reason. **67**(3), 26 (2023)
20. Papadimitriou, C.H.: Games Against Nature. J. Comput. Syst. Sci. **31**(2), 288–301 (1985)
21. Peterson, G., Reif, J., Azhar, S.: Lower bounds for multiplayer noncooperative games of incomplete information. Computers & Mathematics with Applications **41**(7-8), 957–992 (2001)

22. Peterson, G.L., Reif, J.H.: Multiple-Person Alternation. In: FOCS. pp. 348–363. IEEE Computer Society (1979)
23. Phan, Q., Bang, L., Pasareanu, C.S., Malacaria, P., Bultan, T.: Synthesis of Adaptive Side-Channel Attacks. IACR Cryptol. ePrint Arch. p. 401 (2017)
24. Saha, S., Eiers, W., Kadron, I.B., Bang, L., Bultan, T.: Incremental Adaptive Attack Synthesis. CoRR **abs/1905.05322** (2019)
25. Smith, G.: On the Foundations of Quantitative Information Flow. In: FoSSaCS. Lecture Notes in Computer Science, vol. 5504, pp. 288–302. Springer (2009)
26. Vigouroux, T., Ene, C., Monniaux, D., Mounier, L., Potet, M.: BaxMC: a CEGAR approach to Max#SAT. In: FMCAD. pp. 170–178. IEEE (2022)
27. Wang, H., Tu, K., Jiang, J.R., Scholl, C.: Quantifier Elimination in Stochastic Boolean Satisfiability. In: SAT. LIPIcs, vol. 236, pp. 23:1–23:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)
28. Wimmer, R., Scholl, C., Wimmer, K., Becker, B.: Dependency Schemes for DQBF. In: SAT. LNCS, vol. 9710, pp. 473–489. Springer (2016)

## Appendix

*Example 11.* Consider the problem:

$$\max^{\{z_1\}} x_1. \mathbf{R}y_1. \mathbf{R}y_2. \exists z_1. (x_1 \Leftrightarrow y_1) \wedge (z_1 \Leftrightarrow (y_1 \vee y_2))$$

Let  $\Phi_1$  denote the objective formula. As  $\mathcal{F}(\{z_1\}) = \{\top, \perp, z_1, \overline{z_1}\}$  one shall consider these four possible substitutions for the maximizing variable  $x_1$  and compute the associated number of  $\{y_1, y_2\}$ -projected models. For instance,  $\Phi_1[x_1 \mapsto \perp] \equiv \overline{y_1} \wedge (z_1 \Leftrightarrow y_2)$  has two models, respectively  $\{y_1 \mapsto \perp, y_2 \mapsto \top, z_1 \mapsto \top\}$  and  $\{y_1 \mapsto \perp, y_2 \mapsto \perp, z_1 \mapsto \perp\}$  and two  $\{y_1, y_2\}$ -projected models respectively  $\{y_1 \mapsto \perp, y_2 \mapsto \top\}$  and  $\{y_1 \mapsto \perp, y_2 \mapsto \perp\}$ . Therefore  $|\exists z_1. \Phi_1[x_1 \mapsto \perp]|_{\{y_1, y_2\}} = 2$ . The maximizing substitution is  $x_1 \mapsto z_1$  which has three  $\{y_1, y_2\}$ -projected models, that is  $|\exists z_1. \Phi_1[x_1 \mapsto z_1]|_{\{y_1, y_2\}} = 3$ . Note that no substitution for  $x_1$  exists such that the objective to have four  $\{y_1, y_2\}$ -projected models, that is, always valid for counting variables.

*Example 12.* In Program 5, one shall know that the optimal strategy is the dichotomic search of  $y_1$  within its possible values.

```

1  $y_1 \leftarrow \text{random}()$  ;
2  $x_1 \leftarrow \text{input}()$ ;
3  $z_1 \leftarrow \text{output}(x_1 \geq y_1)$ ;
4  $x_2 \leftarrow \text{input}()$ ;
5  $z_2 \leftarrow \text{output}(x_2 \geq y_1)$ ;
6  $x_3 \leftarrow \text{input}()$ ;
7  $\text{win}(x_3 \approx_3^{msb} y_1)$ ;

```

**Program 5:** A second program example