Function synthesis for maximizing model counting

Thomas Vigouroux, Marius Bozga, Cristian Ene, Laurent Mounier

Function synthesis for maximizing model counting

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

Univ. Grenoble Alpes, CNRS, Grenoble INP**, 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 [6,18] 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 the *DQBF* problem [14] as well. 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 on benchmark problems using a prototype *DQMax#SAT* solver implementation.

Keywords: Function synthesis · Model counting · *DQBF* · *Max#SAT* · 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 Φ (capturing some combination of attack objectives). Actually, this corresponds to a classical functional synthesis problem.

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

** Institute of Engineering Univ. Grenoble Alpes

Formulae (*QBF-SAT*) satisfiability problem [7,8] and its generalization, the Dependency Quantified Boolean Formulae (*DQBF*-SAT) satisfiability problem [14].

However, these existing qualitative frameworks are not sufficient in a security context: we are not looking for adversaries able to succeed *in all cases*, but rather for adversaries succeeding with "a good probability". That is why we propose a generalization of the *DQBF* problem: we replace the 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 solutions of Φ, hence the succeeding probability of the attack.

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

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

The problem we aim to solve is then: 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 two well-known existing problems: (i) it generalizes the *Max#SAT* problem [6,18] by allowing the maximizing variables to depend *symbolically* on other variables; (ii) it lifts the *DQBF* problem [14] 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.

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 both the *DQBF* and *Max#SAT* 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* [2] and *programs as information leakage-channels* [17,15] 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* and *DQBF* 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 concludes with some references to related work and proposes some extensions we aim 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 $\alpha_V : V \to \mathbb{B}$ of variables to Boolean values such that ϕ evaluates to $\top$ (that is, *true*) on α_V, it is denoted by $\alpha_V \models \phi$. A formula is satisfiable if it has at least one model α_V. A formula is valid (i.e., tautology) if any assignement α_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 $|\phi|_V \stackrel{def}{=} |\{\alpha_V : V \to \mathbb{B} \mid \alpha_V \models \phi\}|$. 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 $|\exists V_2.\ \phi|_{V_1} \stackrel{def}{=} |\{\alpha_{V_1} : V_1 \to \mathbb{B} \mid \exists \alpha_{V_2} : V_2 \to \mathbb{B}.\ \alpha_{V_1} \uplus \alpha_{V_2} \models \phi\}|$. 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 [4]).

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

2.2 Problem Formulation

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

$$\max^{H_1} x_1.\ ...\ \max^{H_n} x_n.\ \text{Я}Y.\ \exists Z.\ \Phi(X, Y, Z) \tag{1}$$

where $H_1, ..., H_n \subseteq Y \cup Z$ 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 $\sigma^*_X : X \to \mathcal{F}\langle Y \cup Z\rangle$ 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:

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

Let Φ denote the objective formula. In this case, $\mathcal{F}\langle\{z_1, z_2\}\rangle = \{\top, \bot, z_1, \overline{z_1}, z_2, \overline{z_2}, z_1 \vee z_2, \overline{z_1} \vee z_2, z_1 \vee \overline{z_2}, \overline{z_1} \vee \overline{z_2}, z_1 \wedge z_2, \overline{z_1} \wedge z_2, z_1 \wedge \overline{z_2}, \overline{z_1} \wedge \overline{z_2}, z_1 \Leftrightarrow z_2, \overline{z_1 \Leftrightarrow z_2}\}$, 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 ($\{y_1 \mapsto \bot, y_2 \mapsto \top, z_1 \mapsto \top, z_2 \mapsto \bot\}$) 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 σ^* 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:

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

Let Φ denote the associated objective formula. An optimal solution is $x_1 \mapsto \bot, x_2 \mapsto \overline{z_2}$ and one can check that $|\exists z_1.\ \exists z_2.\ \Phi[x_1 \mapsto \bot, x_2 \mapsto \overline{z_2}]|_{\{y_1,y_2\}} = 3$. Moreover, on 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 $y_1 \mapsto \top, y_2 \mapsto \bot$.

2.3 Hardness of *DQMax#SAT*

We briefly discuss now the relationship between the *DQMax#SAT* problem and the well known *Max#SAT* and *DQBF* problems. It turns out that *DQMax#SAT* is harder than both of them, as illustrated by the following rather simple reductions.

***DQMax#SAT* is harder than *Max#SAT*:** Let $X = \{x_1, ..., x_n\}$, Y, Z be pairwise disjoint finite sets of Boolean variables, called respectively *maximizing*, *counting* and *existential* variables. The *Max#SAT* problem [6] specified as

$$\max x_1.\ \ldots \max x_n.\ \text{Ⅎ}Y.\ \exists Z.\ \Phi(X, Y, Z) \tag{2}$$

asks for finding an assignement $\alpha_X^* : X \to \mathbb{B}$ 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, $H_1 = H_2 = ... = H_n = \emptyset$.

***DQMax#SAT* is harder than *DQBF*:** Let $X = \{x_1, ..., x_n\}$, Y be disjoint finite sets of Boolean variables and let $H_1, ..., H_n \subseteq Y$. The *DQBF* problem [14] asks, given a Dependency-Quantified Boolean Formula:

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

to synthesize a substitution $\sigma_X^* : X \to \mathcal{F}\langle Y\rangle$ 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:

$$\max^{H_1} x_1.\ ...\ \max^{H_n} x_n.\ \text{Ⅎ}Y.\ \Phi(X, Y) \tag{4}$$

By solving (4) one can solve the initial *DQBF* problem (3). Indeed, let $\sigma^*_X : X \to \mathcal{F}\langle Y\rangle$ be a solution for (4). Then, the *DQBF* problem admits a solution if and only if $|\Phi[\sigma^*_X]|_Y = 2^{|Y|}$. Moreover, σ^*_X is a solution for the problem (3) because (i) σ^*_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 *DQMax#SAT* to *DQBF*, the maximizing quantifiers in *DQMax#SAT* can be viewed as Henkin quantifiers [10] in *DQBF* with a quantitative flavor.

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_φ of complete monomials from $\mathcal{M}\langle H\rangle$, that is, such that the following equivalences hold:

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

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_φ. 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 define now the following *Max#SAT* problem:

$$(\max x'_{1,m}.)_{m\in\mathcal{M}\langle H_1\rangle} \ldots (\max x'_{n,m}.)_{m\in\mathcal{M}\langle H_n\rangle} \,\text{ЯY}.\, \exists Z.\, \exists X.\\ \Phi(X,Y,Z) \wedge \bigwedge_{i\in[1,n]} \big(x_i \Leftrightarrow \vee_{m\in\mathcal{M}\langle H_i\rangle}(x'_{i,m} \wedge m)\big) \quad (5)$$

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* (5).

Proof. Let us denote

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

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

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

Consequently, finding the substitution σ_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, $\exists Z.\ \exists X.\ \Phi'(X',X,Y,Z)$). □

Example 3. Example 1 is reduced to the following:

$$\begin{aligned}&\max x'_{1,z_1z_2}.\ \max x'_{1,z_1\bar{z_2}}.\ \max x'_{1,\bar{z_1}z_2}.\ \max x'_{1,\bar{z_1}\bar{z_2}}.\ \text{Я}y_1.\ \text{Я}y_2.\ \exists z_1.\ \exists z_2.\ \exists x_1.\\ &(x_1 \Leftrightarrow y_1) \wedge (z_1 \Leftrightarrow y_1 \vee y_2) \wedge (z_2 \Leftrightarrow y_1 \wedge y_2) \wedge\\ &(x_1 \Leftrightarrow \big((x'_{1,z_1z_2} \wedge z_1 \wedge z_2\big) \vee \big(x'_{1,z_1\bar{z_2}} \wedge z_1 \wedge \bar{z_2}\big) \vee \big(x'_{1,\bar{z_1}z_2} \wedge \bar{z_1} \wedge z_2\big) \vee \big(x'_{1,\bar{z_1}\bar{z_2}} \wedge \bar{z_1} \wedge \bar{z_2}\big)))\end{aligned}$$

One possible answer is $x'_{1,z_1z_2} \mapsto \top, x'_{1,z_1\bar{z_2}} \mapsto \top, x'_{1,\bar{z_1}z_2} \mapsto \bot, x'_{1,\bar{z_1}\bar{z_2}} \mapsto \bot$. This yields the solution $\sigma_X(x_1) = (z_1 \wedge z_2) \vee (z_1 \wedge \bar{z_2}) = z_1$ 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 $H'_1 \subseteq H_1, \ldots, H'_n \subseteq H_n$. Actually, if the sets of dependencies $H'_1, \ldots, H'_n$ are chosen such that to augment progressively from $\emptyset, \ldots, \emptyset$ to $H_1, \ldots, H_n$ 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 $\Phi \in \mathcal{F}\langle X \cup Y \cup Z\rangle$, an initial assignment $\alpha_0 : X \to \mathbb{B}$ and a filter formula $\Psi \in \mathcal{F}\langle X\rangle$. 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 α_0 is a model of Ψ and
- forall $\alpha : X \to \mathbb{B}$ if $\alpha \not\models \Psi$ then $|\exists Z.\ \Phi[\alpha]|_Y \leq |\exists Z.\ \Phi[\alpha_0]|_Y$, that is, any assignment α outside of the filter Ψ is at most as good as the assignement α_0.

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

The incremental algorithm proposed in Algorithm 1 proceeds as follows:

input : $X = \{x_1, ..., x_n\}$, Y, Z, H_1, ..., H_n, Φ
output: σ^*_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 $\alpha'_0 \leftarrow \{x'_{i,\top} \mapsto \bot\}_{i\in[1,n]}$
5 $\Psi' \leftarrow \top$
6 **repeat**
7 $\quad \alpha'^* \leftarrow$ `max#sat`$(X', Y, Z \cup X, \Phi', \alpha'_0, \Psi')$
8 $\quad$ **if** $H'_i \neq H_i$ *for some* $i \in [1, n]$ **then**
9 $\quad\quad i_0 \leftarrow$ `choose`$(\{i \in [1, n] \mid H'_i \neq H_i\})$
10 $\quad\quad u \leftarrow$ `choose`$(H_{i_0} \setminus H'_{i_0})$
11 $\quad\quad \alpha'_0 \leftarrow \alpha'^*$
12 $\quad\quad \Psi' \leftarrow \bot$
13 $\quad\quad$ **foreach** $m \in \mathcal{M}\langle H'_{i_0}\rangle$ **do**
14 $\quad\quad\quad X' \leftarrow (X' \setminus \{x'_{i_0,m}\}) \cup \{x'_{i_0,mu}, x'_{i_0,m\bar{u}}\}$
15 $\quad\quad\quad \Phi' \leftarrow \Phi'[x'_{i_0,m} \mapsto (x'_{i_0,mu} \wedge u) \vee (x'_{i_0,m\bar{u}} \wedge \bar{u})]$
16 $\quad\quad\quad \alpha'_0 \leftarrow (\alpha'_0 \setminus \{x'_{i_0,m} \mapsto \_\}) \cup \{x'_{i_0,mu}, x'_{i0,m\bar{u}} \mapsto \alpha'_0(x'_{i_0,m})\}$
17 $\quad\quad\quad \Psi' \leftarrow \Psi' \vee (x'_{i_0,mu} \not\Leftrightarrow x'_{i_0,m\bar{u}})$
18 $\quad\quad$ **end**
19 $\quad\quad \Psi' \leftarrow \Psi' \vee \bigwedge_{x\in X'}(x \Leftrightarrow \alpha'_0(x))$
20 $\quad\quad H'_{i_0} \leftarrow H'_{i_0} \cup \{u\}$
21 $\quad$ **end**
22 **until** $H'_i = H_i$ *for all* $i \in [1, n]$
23 $\sigma^*_X \leftarrow \{x_i \mapsto \vee_{m\in\mathcal{M}\langle H_i\rangle}(\alpha'^*(x'_{i,m}) \wedge m)\}_{i\in[1,n]}$

Algorithm 1: Incremental Algorithm

- at lines 1-5 it prepares the arguments for the first call of the *Max#SAT* oracle, that is, for solving the problem where $H'_1 = H'_2 = ... = H'_n = \emptyset$,
- 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 Φ', it defines the new initial assignement α'_0 and the new filter Ψ' 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 α'^* of the final *Max#SAT* problem where $H'_i = H_i$ for all $i \in [1, n]$ to the corresponding substitution σ^*_X.

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

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

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

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 *DQMax#SAT* problem

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

This is an invariance property provable by induction. It holds by construction of X', Φ', α'_0, Ψ' at the initial step. Then, it is preserved from one oracle call to the next one i.e., X' and Φ' are changed such that to reflect the addition of the variable u of the set H'_{i_0}. The new initial assignement α'_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 assignement α'_0 has exactly the same number of Y-projected models as the optimal assignement α'^* had on the previous problem. The filter Ψ' is built such that to contain this new initial assignment α'_0 (line 19) as well as any other assignement that satisfies $x'_{i_0,mu} \not\Leftrightarrow x'_{i_0,m\bar{u}}$ for some monomial m (lines 12, 17). This construction guarantees that, any assignment which does not satisfy the filter Ψ' reduces precisely to an assignment of the previous problem, other than the optimal one α'^*, and henceforth at most as good as α'_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 (5) and it will therefore allow to derive a correct solution to the initial *DQMax#SAT* problem (1). □

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

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

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

$$\max x'_{1,z_1}.\ \max x'_{1,\bar{z_1}}\ \mathbf{\Re} y_1.\ \mathbf{\Re} y_2.\ \exists z_1.\ \exists z_2.\ \exists x_1. \\ (x_1 \Leftrightarrow y_1) \wedge (z_1 \Leftrightarrow y_1 \vee y_2) \wedge (z_2 \Leftrightarrow y_1 \wedge y_2) \wedge (x_1 \Leftrightarrow x'_{1,z_1} \wedge z_1 \vee x'_{1,\bar{z_1}} \wedge \bar{z_1})$$

A solution found by the oracle is e.g., $x'_{1,z_1} \mapsto \top, x'_{1,\bar{z_1}} \mapsto \bot$ 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, one could in theory stop the search at any given iteration, and construct the returned value σ_X similarly as in Line 23 of Algorithm 1. In this case the returned σ_X would be defined as $\sigma_X = \{x_i \mapsto \vee_{m \in \mathcal{M}\langle H'_i \rangle}(\alpha'^*(x'_{i,m}) \wedge m)\}_{i \in [1,n]}$ (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 Ψ' in Algorithm 1 by solving the *Max#SAT* problem corresponding to $\Phi' \wedge \neg\Psi'$, and return the found solution. Even though the α'_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 Ψ' 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 some counting or existential variable u is occurring in all dependency sets. 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). Let us consider a variable u which occurs in all dependency sets H_i and let $\Phi_u \stackrel{def}{=} \Phi[u \mapsto \top]$, $\Phi_{\bar{u}} \stackrel{def}{=} \Phi[u \mapsto \bot]$ be the two cofactors on variable u of the objective Φ. Let us consider the following u-reduced *DQMax#SAT* problems:

$$\max^{H_1 \setminus \{u\}} x_1. \ldots \max^{H_n \setminus \{u\}} x_n.\ \text{Я}\, Y \setminus \{u\}.\ \exists\, Z \setminus \{u\}.\ \Phi_u \tag{6}$$

$$\max^{H_1 \setminus \{u\}} x_1. \ldots \max^{H_n \setminus \{u\}} x_n.\ \text{Я}\, Y \setminus \{u\}.\ \exists\, Z \setminus \{u\}.\ \Phi_{\bar{u}} \tag{7}$$

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* Φ*,*

then σ^*_X *defined as*

$$\sigma^*_X(x_i) \stackrel{def}{=} \big(u \wedge \sigma^*_{X,u}(x_i)\big) \vee \big(\bar{u} \wedge \sigma^*_{X,\bar{u}}(x_i)\big) \ \textit{for all } i \in [1,n]$$

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 $\varphi_{i,\bar{u}} \stackrel{def}{=} \varphi_i[u \mapsto \bot] \in \mathcal{F}\langle H_i \setminus \{u\} \rangle$. 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 Φ_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 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

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

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

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

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 σ_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 (6) and (7). □

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:

$$\begin{aligned} z_1 \mapsto \top, z_2 \mapsto \top :&\quad \max^{\emptyset} x_1.\ \text{Я}y_1.\ \text{Я}y_2\ .(x_1 \Leftrightarrow y_1) \wedge (\top \Leftrightarrow y_1 \vee y_2) \wedge (\top \Leftrightarrow y_1 \wedge y_2) \\ z_1 \mapsto \top, z_2 \mapsto \bot :&\quad \max^{\emptyset} x_1.\ \text{Я}y_1.\ \text{Я}y_2\ .(x_1 \Leftrightarrow y_1) \wedge (\top \Leftrightarrow y_1 \vee y_2) \wedge (\bot \Leftrightarrow y_1 \wedge y_2) \\ z_1 \mapsto \bot, z_2 \mapsto \top :&\quad \max^{\emptyset} x_1.\ \text{Я}y_1.\ \text{Я}y_2\ .(x_1 \Leftrightarrow y_1) \wedge (\bot \Leftrightarrow y_1 \vee y_2) \wedge (\top \Leftrightarrow y_1 \wedge y_2) \\ z_2 \mapsto \bot, z_2 \mapsto \bot :&\quad \max^{\emptyset} x_1.\ \text{Я}y_1.\ \text{Я}y_2\ .(x_1 \Leftrightarrow y_1) \wedge (\bot \Leftrightarrow y_1 \vee y_2) \wedge (\bot \Leftrightarrow y_1 \wedge y_2) \end{aligned}$$

The four problems are solved independently and have solutions e.g., respectively $x_1 \mapsto c_1 \in \{\top\}$, $x_1 \mapsto c_2 \in \{\top, \bot\}$, $x_1 \mapsto c_3 \in \{\top, \bot\}$, $x_1 \mapsto c_4 \in \{\bot\}$. 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:

$$\begin{aligned}
x_1 &\mapsto (z_1 \wedge z_2 \wedge \top) \vee (z_1 \wedge \overline{z_2} \wedge \bot) \vee (\overline{z_1} \wedge z_2 \wedge \bot) \vee (\overline{z_1} \wedge \overline{z_2} \wedge \bot) && (\equiv z_1 \wedge z_2)\\
x_1 &\mapsto (z_1 \wedge z_2 \wedge \top) \vee (z_1 \wedge \overline{z_2} \wedge \bot) \vee (\overline{z_1} \wedge z_2 \wedge \top) \vee (\overline{z_1} \wedge \overline{z_2} \wedge \bot) && (\equiv z_2)\\
x_1 &\mapsto (z_1 \wedge z_2 \wedge \top) \vee (z_1 \wedge \overline{z_2} \wedge \top) \vee (\overline{z_1} \wedge z_2 \wedge \bot) \vee (\overline{z_1} \wedge \overline{z_2} \wedge \bot) && (\equiv z_1)\\
x_1 &\mapsto (z_1 \wedge z_2 \wedge \top) \vee (z_1 \wedge \overline{z_2} \wedge \top) \vee (\overline{z_1} \wedge z_2 \wedge \top) \vee (\overline{z_1} \wedge \overline{z_2} \wedge \bot) && (\equiv z_1 \vee z_2)
\end{aligned}$$

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.

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 [5,16,15] 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 prevous 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 priviledges on the target platform. Obviously, from the defender side, quantitative measures about the "controlability" 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* [11], 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 rechability property

We show in this section how to encode *quantitative reachability* defined in [2] 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 satisified. 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

```
1 y1 ← random()
2 y2 ← random()
3 z1 ← output(y1 + y2)
4 x1 ← input()
5 if y1 ≤ x1 then
6 |  win(x1 ≤ y2)
7 end
```

Program 2: A first program example

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).

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 at that input time.

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 $z \leftarrow \texttt{output}(e_z)$. Let $X = \{x_1, ..., x_n\}$ be the set of controlled variables with their temporal dependencies respectively subsets $H_1, \ldots, H_n \subseteq Z$ of uncontrollable variables. Finally, let Ψ 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:

$$\max^{H_1} x_1. \ ... \ \max^{H_n} x_n. \ \text{ꓤ}Y. \ \exists Z. \ \Psi \wedge \bigwedge\nolimits_{z \in Z} (z = e_z) \tag{8}$$

```
1 z ← random() //the secret
2 x ← input()
3 if x ≥ z then
4 |  ... some computation taking 10 seconds
5 else
6 |  ... some computation taking 20 seconds
7 end
```

Program 3: A simple leaking program

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

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

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.

6.3 Security as a lack of leakage property

In this section, we extend earlier work on adaptive attackers from [16] 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 [15], the attacker strategy is a concretized tree, which explicitely states, for each concrete program output, what should be the next input provided by the adversary. Following ideas proposed in [15], 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.

Example 8. Let us consider the excerpt Program 3 taken from [15]. 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 [17] states that the information leaked by a program is upper-bounded by the number of different

```
1 z ← random() ;
2 x1 ← input();
3 y1 ← output(x1 ≥ z);
4 x2 ← input();
5 y2 ← output(x2 ≥ z);
6 x3 ← input();
7 y3 ← output(x3 ≥ z);
```

Program 4: An iterated leaking program

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 [15], 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:

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

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 [15] 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 $y \leftarrow \texttt{output}(e_y)$. 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 $z \leftarrow \texttt{random}()$. Let $X = \{x_1, ..., x_n\}$ be the set of controlled variables with their temporal dependencies respectively subsets $H_1, \ldots, H_n \subseteq Y$. Consider now the following most likely generalized *DQMax#SAT* problem:

$$\max^{H_1} x_1.\ ...\ \max^{H_n} x_n.\ \text{Я}Y.\ \exists Z.\ \bigwedge\nolimits_{y \in Y}(y = e_y) \wedge \bigwedge\nolimits_{z \in Z}(z = e_z)$$

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, ..., x_n are input in this order within the program then necessarily $H_1 \subseteq H_2 \subseteq ... \subseteq H_n$. 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. If moreover $H_1 \neq \emptyset$, then this enables the use of the local resolution method described in Section 5. For example, it is the case of Example 6 where z_1 is dependent only on counting variables y_1 and y_2.

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[1] [18] and the exact solver D4max [1].

In the implementation of Algorithm 1 in our tool, the filter Ψ' 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 α_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:

Example 9.

$$\max^{\emptyset} x_1.\ \max^{\{z_1\}} x_2.\ \max^{\{z_1,z_2\}} x_3.\ \text{Я}y_1.\ \exists z_1.\ \exists z_2.\ (x_3 = y_1) \wedge (z_1 = x_1 \geq y_1 \wedge z_2 = x_2 \geq y)$$

Example 10.

$$\max^{\emptyset} x_1.\ \max^{\{y_1\}} x_2.\ \max^{\{y_1,y_2\}} x_3.\ \text{Я}y_1.\ \text{Я}y_2.\ \text{Я}y_3.\ \exists z\ .\ (y_1 \Leftrightarrow x_1 \geq z) \wedge (y_2 \Leftrightarrow x_2 \geq z) \wedge (y_3 \Leftrightarrow x_3 \geq z)$$

[1] Thanks to specific parametrization and the oracles [3] 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

When bitblasting is needed for a given benchmark, the number of bits used for bitblasting is indicated in parentheses.

As you can see in Table 1, the implemented tool can effectively solve all the examples presented in this paper. The synthesized answers returned by both oracles are the same (that is, the selected monomials in Algorithm 1, Line 23 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 [9] for all the synthesized sub-function, before concatenation. This simplification allows us to directly derive the answers explicited in Examples 7 and 8 instead of their equivalent formulated as sums of monomials, and better explain the results returned by the tool.

8 Related work and conclusions

We exposed in this paper a new problem called *DQMax#SAT* that subsumes both *Max#SAT* and *DQBF*. 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.

As demonstrated in Section 2, *DQMax#SAT* subsumes the *DQBF* problem. This relation indicates a similarity of the two problems (one being the quantitative flavor of the other), and thus some related works can be extracted from here. For example, *DQMax#SAT* can be shown to belong to $\mathsf{NEXPTIME}^{\#\mathsf{P}}$ knowing that *DQBF* is NEXPTIME-complete [13] and *Max#SAT* is $\mathsf{NP}^{\#\mathsf{P}}$-complete [18].

Comparing the performances of *DQBF* algorithms with the algorithms we proposed is not yet realistic since they address different objectives. However, thanks to the relationship with *DQBF* and *Max#SAT*, one can search for future directions by looking at the possible optimizations and enhancements arising in these two problems. As summarized in [12], numerous approaches have been proposed to improve the resolution of *DQBF*. For instance, dependency schemes [19] 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. Indeed, any enhancement on the dependencies of any maximizing variables will lead to a significant decrease of the size of the resulting *Max#SAT* problem (5).

From the security point of view, the closest works to our proposal are the ones decribed in [15,16]. 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 [15], the attacker strategy is a concretized tree which explicitely states, for each concrete program output, what should be the next input provided by the adversary.

Our work can be expanded in several directions. First, we would like to enhance our prototype with strategies for dependency expansion (that is, the choice of variable u or the set H_i lines 9 and 10) 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: Meel, K.S., Strichman, O. (eds.) 25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel. LIPIcs, vol. 236, pp. 28:1–28:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.SAT.2022.28, https://doi.org/10.4230/LIPIcs.SAT.2022.28
2. Bardin, S., Girol, G.: A quantitative flavour of robust reachability. CoRR **abs/2212.05244** (2022). https://doi.org/10.48550/arXiv.2212.05244, https://doi.org/10.48550/arXiv.2212.05244
3. Chakraborty, S., Meel, K.S., Vardi, M.Y.: A scalable approximate model counter. In: Schulte, C. (ed.) Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8124, pp. 200–216. Springer (2013). https://doi.org/10.1007/978-3-642-40627-0_18, https://doi.org/10.1007/978-3-642-40627-0_18
4. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Balancing scalability and uniformity in SAT witness generator. CoRR **abs/1403.6246** (2014), http://arxiv.org/abs/1403.6246

5. Dullien, T.: Weird machines, exploitability, and provable unexploitability. IEEE Transactions on Emerging Topics in Computing **8**(2), 391–403 (2020). https://doi.org/10.1109/TETC.2017.2785299
6. Fremont, D.J., Rabe, M.N., Seshia, S.A.: Maximum model counting. In: Singh, S., Markovitch, S. (eds.) Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA. pp. 3885–3892. AAAI Press (2017), http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14968
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
8. Garey, M.R., Johnson, D.S., So, H.C.: An application of graph coloring to printed circuit testing (working paper). In: 16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975. pp. 178–183. IEEE Computer Society (1975). https://doi.org/10.1109/SFCS.1975.3, https://doi.org/10.1109/SFCS.1975.3
9. Gario, M., Micheli, A.: Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In: SMT workshop. vol. 2015 (2015)
10. 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
11. King, J.C.: Symbolic execution and program testing. Communications of the ACM **19**(7), 385–394 (1976)
12. Kovásznai, G.: What is the state-of-the-art in dqbf solving. In: MaCS-16. Joint Conference on Mathematics and Computer Science (2016)
13. 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)
14. Peterson, G.L., Reif, J.H.: Multiple-person alternation. In: 20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979. pp. 348–363. IEEE Computer Society (1979). https://doi.org/10.1109/SFCS.1979.25, https://doi.org/10.1109/SFCS.1979.25
15. Phan, Q., Bang, L., Pasareanu, C.S., Malacaria, P., Bultan, T.: Synthesis of adaptive side-channel attacks. In: 30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017. pp. 328–342. IEEE Computer Society (2017). https://doi.org/10.1109/CSF.2017.8, https://doi.org/10.1109/CSF.2017.8
16. Saha, S., Eiers, W., Kadron, I.B., Bang, L., Bultan, T.: Incremental adaptive attack synthesis. CoRR **abs/1905.05322** (2019), http://arxiv.org/abs/1905.05322
17. Smith, G.: On the foundations of quantitative information flow. In: Foundations of Software Science and Computational Structures: 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings 12. pp. 288–302. Springer (2009)
18. Vigouroux, T., Ene, C., Monniaux, D., Mounier, L., Potet, M.: Baxmc: a CEGAR approach to max#sat. In: Griggio, A., Rungta, N. (eds.) 22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022. pp. 170–178. IEEE (2022). https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_23, https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_23
19. Wimmer, R., Scholl, C., Wimmer, K., Becker, B.: Dependency schemes for DQBF. In: Creignou, N., Berre, D.L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France,

July 5-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9710, pp. 473–489. Springer (2016). https://doi.org/10.1007/978-3-319-40970-2_29, https://doi.org/10.1007/978-3-319-40970-2_29

Appendix

Example 11. Consider the problem:

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

Let Φ_1 denote the objective formula. As $\mathcal{F}\langle\{z_1\}\rangle = \{\top, \bot, z_1, \overline{z_1}\}$ 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, $\Phi_1[x_1 \mapsto \bot] \equiv \overline{y_1} \wedge (z_1 \Leftrightarrow y_2)$ has two models, respectively $\{y_1 \mapsto \bot, y_2 \mapsto \top, z_1 \mapsto \top\}$ and $\{y_1 \mapsto \bot, y_2 \mapsto \bot, z_1 \mapsto \bot\}$ and two $\{y_1, y_2\}$-projected models respectively $\{y_1 \mapsto \bot, y_2 \mapsto \top\}$ and $\{y_1 \mapsto \bot, y_2 \mapsto \bot\}$. Therefore $|\exists z_1.\ \Phi_1[x_1 \mapsto \bot]|_{\{y_1,y_2\}} = 2$. 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 y1 ← random() ;
2 x1 ← input();
3 z1 ← output(x1 ≥ y1);
4 x2 ← input();
5 z2 ← output(x2 ≥ y1);
6 x3 ← input();
7 if x3 ≈3msb y1 then
8 |   win();
9 end
```

Program 5: A second program example