Cobra: A Tool for Solving General Deductive Games

Miroslav Klimoš and Antonín Kučera*

Faculty of Informatics, Masaryk University, Brno, CZ
klimos@mail.muni.cz, kucera@fi.muni.cz

Abstract. We propose a general framework for modelling and solving *deductive games*, where one player selects a secret code and the other player strives to discover this code using a minimal number of allowed experiments that reveal some partial information about the code. The framework is implemented in a software tool Cobra, and its functionality is demonstrated by producing new results about existing deductive games.

1 Introduction

Deductive games (also known as *codebreaking games*) are played by two players, the *codemaker* and the *codebreaker*, where the codemaker selects a secret code from a given finite set, and the codebreaker strives to reveal the code through a series of *experiments* whose outcomes give some partial information about the code. A codebreaker's *strategy* is a recipe how to assemble the next experiment depending on the outcomes of the previous experiments so that the code is eventually discovered. The efficiency of a given strategy is measured either by the maximal number of experiments required to discover the code in the worst case, or by the expected number of experiments required to discover the code assuming the uniform probability distribution over the secret codes. Although various special types of deductive games have been deeply analyzed at both theoretical and experimental level (see below), to the best of authors' knowledge there is *no* software tool which inputs a description of a deductive game (written in a suitable high-level language) and then computes optimal strategies automatically. In this paper, we present a software tool Cobra (COde-BReaking game Analyser [1]) which achieves this functionality. Despite its versatility, Cobra can fully analyze non-trivial deductive games where the number of admissible experiments is very large (10^{64} or even more). Note that one cannot even *enumerate* all of these experiments in reasonable time, and Cobra implements advanced methods for identifying and bypassing families of experiments that are equivalent to already considered ones (up to some symmetry) without considering them explicitly. This is perhaps the most advanced part of Cobra's design which is based on nontrivial concepts and observations (see Section 2). Using Cobra, we were able to produce results about some standard deductive games that were not known before (see Section 3).

Existing works. Simple examples of well-studied deductive games include various board games and puzzles such as *Mastermind* and the *counterfeit coin problem (CCP)*, which are also used as running examples in this paper. In Mastermind, the codemaker chooses a secret sequence of n code pegs of c colors (repetitions allowed).

* Supported by the Czech Science Foundation, grant No. 15-17564S.

The codebreaker tries to reveal the code by making guesses (experiments) which are evaluated by a certain number of black and white markers. A black marker is received for each code peg from the guess which is correct in both color and position. A white marker indicates the existence of a correct color code peg placed in the wrong position. For the classical variant with four pegs and six colors, Knuth [17] demonstrated a strategy that requires five guesses in the worst case and 4.478 guesses on average. Later, Irving [14], Neuwirth [20], and Koyama & Lai [19] presented strategies which improve the expected number of guesses to 4.369, 4.364, and 4.34, respectively (the bound 4.34 is already optimal). More recently, strategies for Mastermind were constructed semi-automatically by using evolutionary algorithms [2], simulated annealing [4], genetic algorithms (see, e.g., [3] and the references therein), or clustering techniques [7].

In the basic variant of the *counterfeit coin problem (CCP)*, one is given N coins, all identical in appearance, and all identical in weight except for one, which is either heavier or lighter than the remaining $N-1$ coins. The goal is to devise a procedure to identify the counterfeit coin using a minimal number of weighings with a balance. This basic variant was considered by Dyson [8] who proved that CCP can be solved with w weighings (experiments) iff $3 \leq N \leq (3^w-3)/2$. There are numerous modifications and generalizations of the basic variant (higher number of counterfeit coins, additional regular coins, multi-pan balance scale, parallel weighing, etc.) which are harder to analyze and in some cases only partial results exist. We refer to [13] for an overview.

Deductive games can also model certain types of attacks in modern security systems based on *information leakage*, where an unauthorized attacker reveals a part of secret information in some unexpected way. For example, in ATM networks, hardware security modules (HSMs) are used to perform sensitive cryptographic operations such as checking a PIN entered by a customer. These HSMs are controlled by a strictly defined API to enforce security. *API-level attacks* are sequences of unanticipated API calls aiming to determine the PIN value; after each call, a piece of information about the PIN value is leaked, and the whole sequence collects enough data to reconstruct the PIN. One such attack, described in [6, 21], can be modeled as a deductive game similar to Mastermind. Clearly, the problem of synthesizing an optimal codebreaker's strategy is highly interesting in this context.

Other examples of deductive games include *string matching games*, where the secret code is a sequence of letters and the codebreaker repeatedly tries to guess the string. Each guess is evaluated by revealing the total number of matching letters. This game was studied already by Erdös & Rényi [10] who gave some asymptotic results about the worst-case number of guesses. Recently, this game found an application in genetics for selecting a subset of genotyped individuals for phenotyping [12, 11].

Due to space constraints, some proofs and tables describing the outcomes of experimental results achieved by Cobra are omitted. These can be found in [16].

2 Cobra: The Underlying Principles

Given a finite or countable set A, the set of all propositional formulae over A is denoted by $\textsc{Form}(A)$. Apart of standard Boolean connectives, we also use the operator $\textsc{exactly}_i$, where $i \in \mathbb{N}$, such that $\textsc{exactly}_i(\varphi_1, \ldots, \varphi_m)$ is true iff exactly i of the formulae $\varphi_1, \ldots, \varphi_m$ are true. For technical convenience, we assume that *all* Boolean connectives used in formulae of $\textsc{Form}(A)$ are commutative. That is, we allow for $\neg, \wedge, \vee, \textsc{exactly}_i, \ldots$, but we forbid implication which must be expressed using the

allowed operators. For a given formula $\varphi \in \textsc{Form}(A)$, we use $\mathit{Val}(\varphi)$ to denote the set of all valuations of A satisfying φ. We write $\varphi \approx \psi$ and $\varphi \equiv \psi$ to denote that φ and ψ are semantically and syntactically equivalent, respectively, and we extend this notation also to sets of formulae. Hence, if Φ, Ψ are sets of formulae, then $\Phi \approx \Psi$ and $\Phi \equiv \Psi$ means that the two sets are the same up to the respective equivalence. The syntactic equivalence $\equiv$ is considered modulo basic identities such as commutativity or associativity.

Our formal model of deductive games is based on propositional logic. Informally, a deductive game is given by

- a finite set X of propositional variables and a propositional formula φ_0 over X such that every secret code c can be represented by a unique valuation v_c of X, and for every valuation v of X we have that $v(\varphi_0) = \mathit{true}$ iff $v = v_c$ for some secret code c;
- a finite set of allowed experiments T.

To model CCP with N coins, we put $X = \{x_1, \ldots, x_N, y\}$, and we represent a secret code c where the i-th coin is heavier by a valuation v_c where $v_c(x_i) = \mathit{true}$, $v_c(x_j) = \mathit{false}$ for all $j \neq i$, and $v_c(y) = \mathit{true}$ (i.e., y is set to *true* iff the different coin is heavier). The formula φ_0 says that precisely one of the variables $x_1, \ldots, x_N$ is set to *true*. In Mastermind with n pegs and m colors, the set X contains variables $x_{i,j}$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$; the variable $x_{i,j}$ is set to *true* iff the i-th peg has color j. The formula φ_0 says that each peg has precisely one color.

Typically, the number of possible experiments is large but many of them differ only in the concrete choice of participating objects. For example, in CCP with 6 coins there are essentially three types of experiments (we can weight either $1+1$, $2+2$, or $3+3$ coins) which are instantiated by a concrete selection of coins. In Mastermind, we perform essentially only one type of experiment (a guess) which is instantiated by a concrete tuple of colors. In general, we use a finite set Σ of *parameters* to represent the objects (such as coins and colors) participating in experiments. A *parameterized experiment* $t \in T$ is a triple (k, P, Φ) where k is the number of parameters, $P \subseteq \Sigma^k$ is the set of admissible instances, and Φ are possible outcomes given as *abstract propositional formulae* (see below).

Definition 1. *A* deductive game *is a tuple* $\mathcal{G} = (X, \varphi_0, \Sigma, F, T)$*, where* X *is a finite set of (propositional) variables,* $\varphi_0 \in \textsc{Form}(X)$ *is a satisfiable* initial constraint*,* Σ *is a finite set of* parameters*, and*

- $F \subseteq X^\Sigma$ *is a set of* attributes *such that for all* $f, f' \in F$ *where* $f \neq f'$ *we have that the images of* f *and* f' *are disjoint,*
- T *is a finite set of* parameterized experiments *of the form* (k, P, Φ) *where* $k \in \mathbb{N}$ *is the number of parameters,* $P \subseteq \Sigma^k$ *is a set of* instances*, and* Φ *is a finite subset of* $\textsc{Form}(X \cup \{f(\$j) \mid f \in F, 1 \leq j \leq k\})$*. The elements of* Φ *are called* outcomes*.*

The intuition behind X, φ_0, and Σ is explained above. Each attribute $f \in F$ corresponds to some "property" that every object $a \in \Sigma$ either does or does not satisfy, and $f(a)$ is the propositional variable of X which encodes the f-property of a. In CCP with N coins, the objects are the coins (i.e., $\Sigma = \{\mathit{coin}_i \mid 1 \leq i \leq N\}$), and for each coin we need to encode the property of "being different". So, there is just one attribute d which maps coin_i to x_i for all $1 \leq i \leq N$. In Mastermind with n pegs and

m colors, each object (color) has the property of "being the color of peg i", where $i \in \{1, \dots, n\}$. Hence, there are n attributes $peg_1, \dots, peg_n$ where $peg_i(color_j) = x_{i,j}$.

Now consider a parameterized experiment $t = (k, P, \Phi)$. An *instance* of t is a k-tuple $\boldsymbol{p} \in P \subseteq \Sigma^k$ of parameters. For every instance $\boldsymbol{p} \in P$ and every outcome $\psi \in \Phi$, we define the $\boldsymbol{p}$*-instance of* ψ as the formula $\psi(\boldsymbol{p}) \in \textsc{Form}(X)$ obtained from ψ by substituting each atom $f(\$j)$ with the variable $f(\boldsymbol{p}_j)$. Hence, $f(\$j)$ denotes the variable which encodes the f-attribute of $\boldsymbol{p}_j$. In the rest of this paper, we typically use φ, ψ to range over outcomes, and ξ, χ to range over their instances. We also use E to denote the set of all *experiment instances* (or just *experiments*) defined by $E = \{(t, \boldsymbol{p}) \mid t \in T,\ \boldsymbol{p} \text{ is an instance of } t\}$. Further, for every experiment $e = (t, \boldsymbol{p})$, we use $\Phi(e)$ to denote the set of $\boldsymbol{p}$-instances of all outcomes of t. An *evaluated experiment* is a pair (e, ξ), where $\xi \in \Phi(e)$. The set of all evaluated experiments is denoted by Ω.

Example 2. CCP with four coins can be modeled as a game $\mathcal{G} = (X, \varphi_0, \Sigma, F, T)$ where $X = \{x_1, x_2, x_3, x_4, y\}$, $\varphi_0 = \textsc{exactly}_1(x_1, x_2, x_3, x_4)$, $\Sigma = \{coin_1, coin_2, coin_3, coin_4\}$, $F = \{d\}$ where $d(coin_i) = x_i$ for every $1 \leq i \leq 4$, and $T = \{t_1, t_2\}$ where $t_1 = (2, \Sigma^{\langle 2 \rangle}, \{\varphi_<, \varphi_=, \varphi_>\})$, $t_2 = (4, \Sigma^{\langle 4 \rangle}, \{\psi_<, \psi_=, \psi_>\})$, and

$$
\begin{aligned}
\varphi_< &= (d(\$1) \wedge \neg y) \vee (d(\$2) \wedge y) \\
\varphi_= &= \neg d(\$1) \wedge \neg d(\$2) \\
\varphi_> &= (d(\$1) \wedge y) \vee (d(\$2) \wedge \neg y) \\
\psi_< &= ((d(\$1) \vee d(\$2)) \wedge \neg y) \vee ((d(\$3) \vee d(\$4)) \wedge y) \\
\psi_= &= \neg d(\$1) \wedge \neg d(\$2) \wedge \neg d(\$3) \wedge \neg d(\$4) \\
\psi_> &= ((d(\$1) \vee d(\$2)) \wedge y) \vee ((d(\$3) \vee d(\$4)) \wedge \neg y)
\end{aligned}
$$

Here, $\Sigma^{\langle k \rangle} \subseteq \Sigma^k$ consists of all $w \in \Sigma^k$ such that every letter of Σ appears at most once in w. Note that t_1 and t_2 correspond to weighings of $1 + 1$ and $2 + 2$ coins, respectively. The formulae $\varphi_<$, $\varphi_=$, and $\varphi_>$ encode the three possible outcomes of weighing $1 + 1$ coins. In particular, $\varphi_<$ describes the outcome when the left pan is lighter; then we learn that either the first coin is different and lighter, or the second coin is different and heavier. If we put $\boldsymbol{p} = (coin_4, coin_3)$, then $\varphi_<(\boldsymbol{p})$ is the formula $(x_4 \wedge \neg y) \vee (x_3 \wedge y)$.

For the rest of this section, we fix a deductive game $\mathcal{G} = (X, \varphi_0, \Sigma, F, T)$. We assume that $\mathcal{G}$ is *well-formed*, i.e., for every valuation of $Val(\varphi_0)$, each experiment produces exactly one valid outcome (deductive games that correctly encode meaningful problems are well-formed, so this condition is not restrictive). Intuitively, the game $\mathcal{G}$ is played as follows:

1. The codemaker selects a secret code $v \in Val(\varphi_0)$.
2. The codebreaker selects the next experiment $e \in E$.
3. The codemaker evaluates e for v and returns the resulting evaluated experiment (e, ξ).
4. If the codemaker has enough information to determine v, the play ends. Otherwise, it continues with Step 2.

We assume that the only information available to the codebreaker is the history of evaluated experiments played so far. Hence, a *strategy* is a (total) function $\sigma : \Omega^* \to E$ which specifies the next experiment for a given finite history of evaluated experiments.

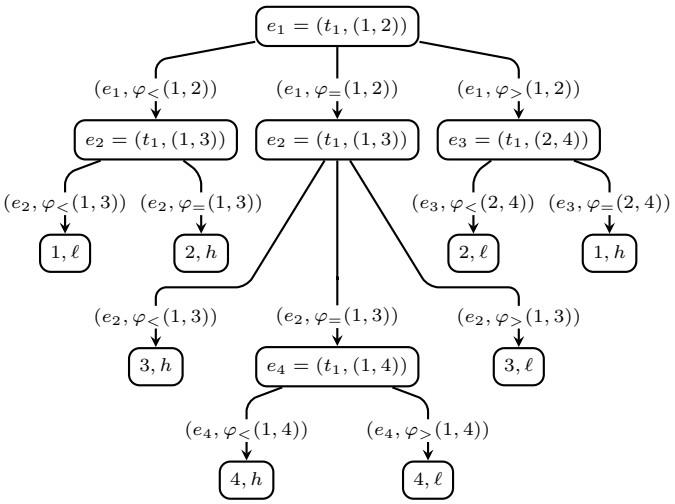


Fig. 1. A decision tree for a simple strategy.

Every strategy σ determines the associated *decision tree*, denoted by Tree_σ, where the internal nodes are labeled by experiments, the leaves are labeled by valuations of $\mathit{Val}(\varphi_0)$, and the edges are labeled by evaluated experiments. For every node u of Tree_σ, let $\lambda^\sigma_u = (e_1, \xi_1), \ldots, (e_n, \xi_n)$ be the unique sequence of evaluated experiments that label the edges of the unique finite path from the root of Tree_σ to u (note that if u is the root, then $\lambda^\sigma_u = \varepsilon$). We also use Ψ^σ_u to denote the formula $\varphi_0 \wedge \xi_1 \wedge \cdots \wedge \xi_n$. The structure of Tree_σ is completely determined by the following conditions:

- Every node u of Tree_σ is either an internal node labeled by $\sigma(\lambda^\sigma_u)$, or a leaf labeled by the only valuation of $\mathit{Val}(\Psi^\sigma_u)$, depending on whether $|\mathit{Val}(\Psi^\sigma_u)| > 1$ or not, respectively.
- Every internal node u of Tree_σ labeled by e has a unique successor u_ξ for each $\xi \in \Phi(e)$ such that the formula $\Psi^\sigma_u \wedge \xi$ is still satisfiable. The edge from u to u_ξ is labeled by (e, ξ).

Note that different nodes/edges may have the same labels, and Tree_σ may contain infinite paths in general.

Example 3. Consider the game $\mathcal{G}$ of Example 2. A decision tree for a simple strategy σ is shown in Fig. 1 (we write just i instead of coin_i, and we use i, ℓ (or i, h) to denote the valuation of $\mathit{Val}(\varphi_0)$ which sets x_i to *true* and y to *false* (or to *true*, respectively)). Note that σ discovers the secret code by performing at most three experiments. Also note that some internal nodes have only two successors, because the third outcome is impossible.

Since $\mathcal{G}$ is well-formed, every strategy σ and every $v \in \mathit{Val}(\varphi_0)$ determine a unique (finite or infinite) path $u_1, u_2, u_3, \ldots$ initiated in the root of Tree_σ, which intuitively correspond to a *play* of $\mathcal{G}$ where the codemaker selects the secret code v. We use $\lambda^\sigma_v = (e_1, \xi_1), (e_2, \xi_2), (e_3, \xi_3), \ldots$ to denote the associated sequence of evaluated experiments, i.e., (e_i, ξ_i) is the label of (u_i, u_{i+1}). The length of λ^σ_v is denoted by

$\#\lambda^\sigma_v$. Further, for every $k \leq \#\lambda^\sigma_v$, we use $\Psi^\sigma_v[k]$ to denote the formula $\Psi^\sigma_{u_k}$ which represents the knowledge accumulated after evaluating the first k experiments.

Now we can define the *worst/average case complexity of* σ, denoted by $\mathcal{C}_{worst}(\sigma)$ and $\mathcal{C}_{avg}(\sigma)$, in the following way:

$$\mathcal{C}_{worst}(\sigma) = \max\{\#\lambda^\sigma_v \mid v \in \mathit{Val}(\varphi_0)\} \qquad \mathcal{C}_{avg}(\sigma) = \frac{\sum_{v \in \mathit{Val}(\varphi_0)} \#\lambda^\sigma_v}{|\mathit{Val}(\varphi_0)|}$$

Note that the worst/average case complexity of σ is finite iff *every* $v \in \mathit{Val}(\varphi_0)$ is discovered by σ after a finite number of experiments. We say that $\mathcal{G}$ is *solvable* iff there exists a strategy σ with a finite worst/average case complexity. Further, we say that a strategy σ is *worst case optimal* iff for every strategy σ' we have that $\mathcal{C}_{worst}(\sigma) \leq \mathcal{C}_{worst}(\sigma')$. Similarly, σ is *average case optimal* iff $\mathcal{C}_{avg}(\sigma) \leq \mathcal{C}_{avg}(\sigma')$ for every strategy σ'.

In general, a codebreaker's strategy may depend not only on the outcomes of previously evaluated experiments, but also on their order. Now we show that the codebreaker can select the next experiment *only* according to the semantics of the knowledge accumulated so far.

Definition 4. *A strategy* σ *is* knowledge-based *if for all* $v_1, v_2 \in \mathit{Val}(\varphi_0)$ *and* $k_1, k_2 \in \mathbb{N}$ *such that* $\Psi^\sigma_{v_1}[k_1] \approx \Psi^\sigma_{v_2}[k_2]$ *we have that* $\sigma(\lambda^\sigma_{v_1}(1), \ldots, \lambda^\sigma_{v_1}(k_1)) = \sigma(\lambda^\sigma_{v_2}(1), \ldots, \lambda^\sigma_{v_2}(k_2))$.

The next theorem says that knowledge-based strategies are equally powerful as general strategies.

Theorem 5. . *Let* $\mathcal{G}$ *be a well-formed deductive game. For every strategy* σ *there exists a knowledge-based strategy* τ *such that for every* $v \in \mathit{Val}(\varphi_0)$ *we have that* $\#\lambda^\tau_v \leq \#\lambda^\sigma_v$.

In the proof of Theorem 5, we show that the only reason why σ might *not* be knowledge-based is that σ schedules completely useless experiments which can be safely omitted. Thus, we transform σ into τ.

Since the codebreaker may safely determine the next experiment just by considering the currently accumulated knowledge, we can imagine that he somehow "ranks" the outcomes of available experiments and then chooses the most promising one. More precisely, let $\textsc{Know} \subseteq \textsc{Form}(X)$ be the set of all formulae representing an accumulated knowledge, i.e., $\textsc{Know}$ consists of all $\Psi^\sigma_v[k]$ where σ is a strategy, $v \in \mathit{Val}(\varphi_0)$, and $k \in \mathbb{N}$. For every $\varphi \in \textsc{Know}$ and every experiment $e \in E$, we define the set

$$\mathit{Updates}[\varphi, e] = \{\varphi \wedge \xi \mid \xi \in \Phi(e)\}$$

which represents possible "updates" in the accumulated knowledge that can be obtained by performing e. Further, let $r : 2^{\textsc{Know}} \to \mathbb{R}$ be a *ranking function*, and $\preceq$ (some) total ordering over the set E of all experiments.

Definition 6. *A* ranking strategy determined by r and $\preceq$ *is a function* $\tau[r, \preceq] : \textsc{Know} \to E$ *such that* $\tau[r, \preceq](\varphi)$ *is the least element of* $\{e \in E \mid r(\mathit{Updates}[\varphi, e]) = \mathit{Min}\}$ *w.r.t.* $\preceq$, *where* $\mathit{Min} = \min\{r(\mathit{Updates}[\varphi, e']) \mid e' \in E\}$.

Note that every ranking strategy can be understood as a "general" strategy, and hence all notions introduced for general strategies (such as the decision tree) make sense also for ranking strategies. Further, for every knowledge-based strategy τ there is an "equivalent" ranking strategy $\tau[r, \preceq]$ where, for all $\varphi \in \textsc{Know}$ and $e \in E$, the value of $r(\mathit{Updates}[\varphi, e])$ is either 0 or 1, depending on whether $\mathit{Updates}[\varphi, e]$ is equal to $\mathit{Updates}[\varphi, \tau(\varphi)]$ or not, respectively. The ordering $\preceq$ can be chosen arbitrarily. One can easily show that for every $v \in \mathit{Val}(\varphi_0)$ we have that $\#\lambda^{\tau}_{v} = \#\lambda^{\tau[r,\preceq]}_{v}$. So, ranking strategies are equally powerful as knowledge-based strategies and hence also general strategies by Theorem 5. In particular, there exist worst/average case optimal ranking strategies, but it is not clear what kind of ranking functions they need to employ. Since optimal strategy synthesis is computationally costly, one may also *fix* some r and $\preceq$, synthesize $\tau[r, \preceq]$, and evaluate its worst/average case complexity. Thus, by experimenting with different r and $\preceq$, one may obtain various strategies that solve the game, and then choose the most efficient one.

Now we introduce several distinguished ranking functions (all of them are implemented in Cobra). They generalize concepts previously used for solving Mastermind, and there are also two new rankings based on the number of fixed variables. The associated ranking strategies always use the lexicographical ordering over E determined by some fixed linear orderings over the sets T and Σ.

- **max-models**$(\Psi) = \max_{\psi \in \Psi} |\mathit{Val}(\psi)|$. The associated ranking strategy minimizes the worst-case number of remaining secret codes. For Mastermind, this was suggested by Knuth [17].
- **exp-models**$(\Psi) = \frac{\sum_{\psi \in \Psi} |\mathit{Val}(\psi)|^2}{\sum_{\psi \in \Psi} |\mathit{Val}(\psi)|}$. The associated ranking strategy minimizes the expected number of remaining secret codes. For Mastermind, this was suggested by Irwing [14].
- **ent-models**$(\Psi) = \sum_{\psi \in \Psi} \frac{|\mathit{Val}(\psi)|}{N} \cdot \log(\frac{|\mathit{Val}(\psi)|}{N})$, where $N = \sum_{\psi \in \Psi} |\mathit{Val}(\psi)|$. The associated ranking strategy minimizes the entropy of the numbers of remaining secret codes. For Mastermind, this was suggested by Neuwirth [20].
- **parts**$(\Psi) = -|\{\psi \in \Psi \mid \psi \text{ is satisfiable}\}|$. The associated ranking strategy minimizes the number of satisfiable outcomes. For Mastermind, this was suggested by Kooi [18].

We say that a variable $x \in X$ is *fixed* in a formula $\varphi \in \textsc{Form}(X)$ if x is set to the same value by all valuations satisfying φ (i.e., for all $v, v' \in \mathit{Val}(\varphi)$ we have that $v(x) = v'(x)$). The set of all variables that are fixed in φ is denoted by $\mathit{Fix}(\varphi)$. We consider two ranking functions based on the number of fixed variables.

- **min-fixed**$(\Psi) = -\min_{\psi \in \Psi} |\mathit{Fix}(\psi)|$. The associated ranking function maximizes the number of fixed variables.
- **exp-fixed**$(\Psi) = -\frac{\sum_{\psi \in \Psi} |\mathit{Val}(\psi)| \cdot |\mathit{Fix}(\psi)|}{\sum_{\psi \in \Psi} |\mathit{Val}(\psi)|}$. The associated ranking function maximizes the expected number of fixed variables.

Intuitively, a "good" ranking function should satisfy two requirements:

- The associateted ranking strategy should have a low worst/average case complexity. Ideally, this strategy should be optimal.
- The ranking function should be easy to evaluate for a given experiment e. This is crucial for automatic strategy synthesis.

	max-models	**exp-models**	**ent-models**	**parts**	**min-fixed**	**exp-fixed**
e_1	4	3	-1.04	-3	-2	-2
e_2	4	4	-0.69	-2	0	0

Table 1. A table summarizing the outcomes of ranking functions.

Obviously, there is a conflict in these two requirement. For example, the **max-models** ranking often produces a rather efficient strategy, but the number of satisfying valuations of a given propositional formula is hard to compute. On the other hand, **min-fixed** ranking produces a good ranking strategy only in some cases (e.g., for CCP and its variants), but it is relatively easy to compute with modern SAT solvers even for large formulae.

Example 7. Consider again the game $\mathcal{G}$ of Example 2 formalizing CCP with four coins. Further, consider the experiments

- $e_1 = (t_1, (\mathit{coin}_1, \mathit{coin}_2))$,
- $e_2 = (t_2, (\mathit{coin}_1, \mathit{coin}_2, \mathit{coin}_3, \mathit{coin}_4))$

for the first step (i.e., when the current accumulated knowledge is just φ_0). In e_1, we weight coin_1 against coin_2. The number of satisfying assignments is 2 for the outcomes $\varphi_<$ and $\varphi_>$, and 4 for the outcome $\varphi_=$. For the outcomes $\varphi_<$ and $\varphi_>$, we know that the counterfeit coin is *not* among coin_3 and coin_4, and for the $\varphi_=$ outcome, we know it is not among coin_1 and coin_2. Hence, every outcome fixes 2 variables. Similarly, we can evaluate e_2 and the other ranking functions. The results are summarized in Table 1. Observe that all of the considered ranking strategies would prefer e_1 to e_2 in the first step, possibly except for the **max-models** ranking strategy where the choice depends on the chosen liner ordering over T and Σ (it t_1 is smaller that t_2, this strategy also prefers e_1).

Although computing $\tau[r, \preceq]$ for given r and $\preceq$ appears computationally easier than synthesizing an optimal strategy, we still need to (repeatedly) compute the least element of $\{e \in E \mid r(\mathit{Updates}[\varphi, e]) = \mathit{Min}\}$ w.r.t. $\preceq$, where $\mathit{Min} = \min\{r(\mathit{Updates}[\varphi, e']) \mid e' \in E\}$, which is not achievable by enumerating all experiments. For example, in CCP with 60 coins, there are more than 10^{63} ways of instantiating the parameterized experiment t formalizing the weighing of $20+20$ coins. However, observe that if t is performed in the first step, i.e., when the accumulated knowledge is just φ_0, then *all* instances of t are "equivalent" in the sense that the knowledge learned by these instances is the same up to a permutation of coins. Hence, it suffices to consider only *one* instance of t and disregard the others. Cobra implements an algorithm which can efficiently recognize and exploit such symmetries. Now we briefly explain the main ideas behind this algorithm.

A *permutation* of X is a bijection $\pi : X \to X$. We use $\textsc{Perm}(X)$ to denote the set of all permutations of X. Given a formula $\varphi \in \textsc{Form}(X)$ and a permutation $\pi \in \textsc{Perm}(X)$, we use $\pi(\varphi)$ to denote the formula obtained from φ by simultaneously substituting every occurrence of every $x \in X$ with $\pi(x)$. For a given $\Phi \subseteq \textsc{Form}(X)$, we use $\pi(\Phi)$ to denote the set $\{\pi(\varphi) \mid \varphi \in \Phi\}$.

Definition 8. *Let $e, e' \in E$ and $\pi \in \textsc{Perm}(X)$. We say that e' is* π-symmetrical *to e if $\pi(\Phi(e)) \approx \Phi(e')$. A* symmetry group of $\mathcal{G}$*, denoted by Π, consist of all $\pi \in \textsc{Perm}(X)$ such that for every $e \in E$ there is a π-symmetrical $e' \in E$.*

We say that $e, e' \in E$ *are* equivalent w.r.t. a given $\varphi \in \textsc{Know}$*, written* $e \sim_\varphi e'$*, if there is* $\pi \in \Pi$ *such that* $\{\varphi \wedge \psi \mid \psi \in \Phi(e)\} \approx \{\pi(\varphi \wedge \varrho) \mid \varrho \in \Phi(e')\}$.

Note that Π is indeed a group, i.e., Π contains the identity and if $\pi \in \Pi$, then the inverse π^{-1} of π also belongs to Π.

Example 9. Consider the game of Example 2. Then $\Pi = \{\pi \in \textsc{Perm}(X) \mid \pi(y) = y\}$. Hence, for all $\boldsymbol{p}, \boldsymbol{q} \in \Sigma^{\langle 4 \rangle}$ we have that $(t_2, \boldsymbol{p}) \sim_{\varphi_0} (t_2, \boldsymbol{q})$, and the partition $E/\!\sim_{\varphi_0}$ has only two equivalence classes corresponding to t_1 and t_2. For $\varphi = \varphi_0 \wedge \neg(x_1 \vee x_2)$, we have that $(t_1(\mathit{coin}_4, \mathit{coin}_3)) \sim_\varphi (t_2, (\mathit{coin}_3, \mathit{coin}_1, \mathit{coin}_2, \mathit{coin}_4))$.

The core of Cobra are the algorithms for synthesizing worst/average case optimal strategies, and for analyzing the efficiency of $\tau[r, \preceq]$. For a current accumulated knowledge $\varphi \in \textsc{Know}$, these algorithms need to consider at least one experiment for each equivalence class of $E/\!\sim_\varphi$. This is achieved by invoking a function $\textsc{Experiments}(\varphi)$ parameterized by φ which computes a set of experiments $S_\varphi \subseteq E$ such that for every $e \in E$ there is at least one $e' \in S_\varphi$ where $e \sim_\varphi e'$. A naive approach to constructing S_φ is to initialize $\hat{S}_\varphi := \emptyset$ and then process every $t = (k, P, \Phi) \in T$ as follows: for every $\boldsymbol{p} \in \Sigma^k$, we check whether $\boldsymbol{p} \in P$ and $(t, \boldsymbol{p}) \not\sim_\varphi e$ for all $e \in \hat{S}_\varphi$; if this test is positive, we put $\hat{S}_\varphi := \hat{S}_\varphi \cup \{(t, \boldsymbol{p})\}$, and continue with the next $\boldsymbol{p}$. When we are done with all $t \in T$, we set $S_\varphi := \hat{S}_\varphi$. Obviously, this trivial algorithm is inefficient for at least two reasons.

1. The size of Σ^k can be very large (think again of CCP with 60 coins), and it may not be possible to go over all $\boldsymbol{p} \in \Sigma^k$.
2. The problem of checking $\sim_\varphi$ is computationally hard.

Now we indicate how Cobra overcomes these issues. Intuitively, the first issue is tackled by optimizing the trivial backtracking algorithm which would normally generate all elements of Σ^k lexicographically using some total ordering $\preceq$ over Σ. We improve the functionality of this algorithm as follows: when the backtracking algorithm is done with generating all k-tuples starting with a given prefix $\boldsymbol{u}a \in \Sigma^m$, where $m \in \{1, \ldots, k\}$, and aims to generate all k-tuples starting with $\boldsymbol{u}b$, we first check whether $\boldsymbol{u}b$ is *dominated* by $\boldsymbol{u}a$ w.r.t. φ and t. The dominance by $\boldsymbol{u}a$ guarantees that all of the experiments that would be obtained by using the k-tuples starting with $\boldsymbol{u}b$ are equivalent to some of the already generated ones. Hence, if $\boldsymbol{u}b$ is dominated by $\boldsymbol{u}a$ w.r.t. φ and t, we continue immediately with the $\preceq$-successor c of b, i.e., we do *not* examine the k-tuples starting with $\boldsymbol{u}b$ at all (note that $\boldsymbol{u}c$ is again checked for dominance by $\boldsymbol{u}a$). This can lead to drastic improvements in the total number of generated instances which can be *much* smaller than $|\Sigma|^k$. The set of all experiments generated in the first phase is denoted by S^1_φ.

The second issue is tackled by designing an algorithm which tries to decide $\sim_\varphi$ for a given pair of experiments e_1, e_2 by first removing the *fixed variables* in φ and the outcomes of e_1, e_2 using a SAT solver, and then constructing two labeled graphs B_{φ, e_1} and B_{φ, e_1} which are checked for isomorphism (here Cobra relies on existing software tools for checking graph isomorphism). If the graphs are isomorphic, we have that $e_1 \sim_\varphi e_2$, and we can safely remove e_1 or e_2 from S^1_φ. When the experiments are ordered by some $\preceq$, we prefer to remove the larger one. Thus, we produce the set S_φ. Now we explain both phases in greater detail.

Let $t = (k, P, \Phi)$ be a parameterized experiment, and let $i, j \in \{1, \ldots, k\}$ be two positions. We say that i, j are *closely dependent* if $i = j$ or there exists an

attribute $f \in F$ such that both $f(\$i)$ and $f(\$j)$ occur in the formulae of Φ. Further, we say that i, j are *dependent* if they are related by the transitive closure of close dependence relation. Note that the set $\{1, \ldots, k\}$ can be partitioned into disjoint subsets of mutually dependent indexes. Further, for every $i \in \{1, \ldots, k\}$ we define the set F_i consisting of all $f \in F$ such that $f(\$j)$ occurs in some formula of Φ where $j \in \{1, \ldots, k\}$ and i, j are dependent.

As an example, consider the parameterized experiment t_2 in the game of Example 2. Then all indexes are mutually dependent and $F_i = \{d\}$ for every $i \in \{1, 2, 3, 4\}$. In Mastermind with n pegs and m colors, there is only one parameterized experiment $t = (n, \{color_1, \ldots, color_m\}^n, \Phi)$, and all indexes are again mutually dependent. We have that $F_i = \{peg_1, \ldots, peg_n\}$ for all $i \in \{1, \ldots, n\}$.

We say that $\boldsymbol{r} \in \Sigma^i$, where $1 \leq i \leq k$, is *t-feasible* if there is $\boldsymbol{s} \in \Sigma^{k-i}$ such that $\boldsymbol{rs} \in P$. Further, for all $\boldsymbol{p} \in \Sigma^k$, $m \in \{1, \ldots, k\}$, and $a, b \in \Sigma$, we denote by $\boldsymbol{p}[m, a{\leftrightarrow}b]$ the element of Σ^k obtained from p by simultaneously substituting every occurrence of a with b and every occurrence of b with a at all positions j where m and j are dependent.

Definition 10. *Let $\varphi \in$ KNOW, $t = (k, P, \Phi) \in T$, and let $\boldsymbol{u}a \in \Sigma^m$ be a t-feasible tuple, where $1 \leq m < k$. We say that $\boldsymbol{u}b \in \Sigma^m$ is* dominated *by $\boldsymbol{u}a$ w.r.t. φ and t if the following conditions are satisfied:*

- *for every $\boldsymbol{v}$ where $\boldsymbol{p} = \boldsymbol{u}b\boldsymbol{v} \in P$ we have that $\boldsymbol{p}[m, a{\leftrightarrow}b] \in P$ and $\boldsymbol{p}[m, a{\leftrightarrow}b] \preceq \boldsymbol{p}$;*
- *for every $f \in F_m$, the variables $f(a)$ and $f(b)$ do not occur in the formulae of Φ;*
- *the permutation π, defined by $\pi(f(a)) = f(b)$, $\pi(f(b)) = f(a)$ for all $f \in F_m$, and $\pi(y) = y$ for the other variables, is a symmetry of φ, i.e., $\varphi \equiv \pi(\varphi)$.*

Theorem 11. *Let $\varphi \in$ KNOW, $t = (k, P, \Phi) \in T$, and let $\boldsymbol{u}a \in \Sigma^m$ be a t-feasible tuple, where $1 \leq m < k$. If $\boldsymbol{u}b$ is dominated by $\boldsymbol{u}a$ w.r.t. φ and t, then for every $\boldsymbol{v} \in \Sigma^{k-m}$ such that $\boldsymbol{p} = \boldsymbol{u}b\boldsymbol{v} \in P$ we have that $\boldsymbol{p}[m, a{\leftrightarrow}b] \in P$ and $(t, \boldsymbol{p}) \sim_\varphi (t, \boldsymbol{p}[m, a{\leftrightarrow}b])$.*

Proof. Let $\boldsymbol{q} = \boldsymbol{p}[m, a{\leftrightarrow}b]$, and let π be the permutation introduced in Definition 10. We show that $\{\varphi \wedge \psi \mid \psi \in \Phi((t, \boldsymbol{p}))\} \equiv \{\pi(\varphi \wedge \varrho) \mid \varrho \in \Phi((t, \boldsymbol{q}))\}$. Since $\varphi \equiv \pi(\varphi)$, it suffices to prove that $\Psi(\boldsymbol{p}) \equiv \pi(\Psi(\boldsymbol{q}))$ for all $\Psi \in \Phi$. Let us fix some $\Psi \in \Phi$. Observe that the formulae $\Psi(\boldsymbol{p})$ and $\pi(\Psi(\boldsymbol{q}))$ are the same except that all $f(\$i)$ are evaluated either to $f(\boldsymbol{p}_i)$ or to $\pi(f(\boldsymbol{q}_i))$, respectively. Let us examine possible cases.

- If $a \neq \boldsymbol{p}_i \neq b$, then $\boldsymbol{p}_i = \boldsymbol{q}_i$ and $\pi(f(\boldsymbol{q}_i)) = \pi(f(\boldsymbol{p}_i)) = f(\boldsymbol{p}_i)$ by Definition 10.
- If i and m are independent, then again $\boldsymbol{p}_i = \boldsymbol{q}_i$ and $\pi(f(\boldsymbol{q}_i)) = \pi(f(\boldsymbol{p}_i)) = f(\boldsymbol{p}_i)$ by Definition 10 (note that $f \notin F_m$).
- If i, m are dependent and $\boldsymbol{p}_i = a$, then $\pi(f(\boldsymbol{p}_i)) = \pi(f(a)) = f(b) = f(\boldsymbol{q}_i)$ because $f \in F_i$. The case when i, m are dependent and $\boldsymbol{p}_i = b$ is symmetric. □

Theorem 11 fully justifies the correctness of the improved backtracking algorithm discussed above in the sense that the resulting set S^1_φ indeed contains at least one representative for each equivalence class of $E/{\sim_\varphi}$.

Now we describe the second phase, when we try to identify and remove some equivalent experiments in S^1_φ. The method works only under the condition that for every $t = (k, P, \Phi) \in T$ we have that P is closed under all permutations of Σ (note that this condition is satisfied when $P = \Sigma^k$ or $P = \Sigma^{\langle k \rangle}$). Possible generalizations are left for future work. The method starts by constructing a labeled *base graph* $B = (V, E, L)$ of $\mathcal{G}$, where the set of vertices V is $X \cup F$ (we assume $X \cap F = \emptyset$) and the edges of E are determined as follows:

– $(f, x) \in E$, where $f \in F$ and $x \in X$, if there is $a \in \Sigma$ such that $f(a) = x$;
– $(x, y) \in E$, where $x, y \in X$, if there are $a \in \Sigma$, $f, g \in F$, $t \in T$, some outcome ψ of T, such that $f(a) = x$, $g(a) = y$, and both $f(\$i)$ and $g(\$i)$ appear in ψ for some $i \in \{1, \ldots, k\}$.

The labeling $L : V \to X \cup F \cup \{var\}$, where $var \notin X \cup F$, assigns var to every variable $x \in X$ such that x does not appear in any outcome of any parameterized experiment of T. For the other vertices $v \in V$, we have that $L(v) = v$. The base graph B represents a subset of Π in the following sense:

Theorem 12. *Let π be an automorphism of B. Then π restricted to X belongs to Π.*

Now, let $\varphi \in \textsc{Form}_X$ be a formula representing the accumulated knowledge, and let $e_1 = (t_1, \boldsymbol{p})$ and $e_2 = (t_2, \boldsymbol{q})$ be experiments. We show how to construct two labeled graphs B_{φ,e_1} and B_{φ,e_2} such that the existence of an isomorphism between B_{φ,e_1} and B_{φ,e_2} implies $e_1 \sim_\varphi e_2$.

For every formula $\psi \in \textsc{Form}_X$, let $\mathit{Stree}(\psi)$ be the syntax tree of ψ, where every inner node is labeled by the associated Boolean operator, the leaves are labeled by the associated variables of X, and the root is a fresh vertex $\mathit{root}(\psi)$ with only one successor which corresponds to the topmost operator of ψ (the label of $\mathit{root}(\psi)$ is irrelevant for now). Recall that we only allow for commutative operators, so the ordering of successors of a given inner node of $\mathit{Stree}(\psi)$ is not significant. Each such $\mathit{Stree}(\psi)$ can be *attached* to any graph B' which subsumes B by taking the disjoint union of the vertices of B' and the inner vertices of $\mathit{Stree}(\psi)$, and identifying all leaves of $\mathit{Stree}(\psi)$ labeled by $x \in X$ with the unique node x of B'. All edges and labels are preserved.

The graph B_{φ,e_1} is obtained by subsequently attaching the formulae $\mathit{Stree}(\overline{\varphi})$, $\mathit{Stree}(\overline{\psi_1(\boldsymbol{p})}), \ldots, \mathit{Stree}(\overline{\psi_n(\boldsymbol{p})})$ to the base graph of B, where $\psi_1, \ldots, \psi_n$ are the outcomes of t_1, and for every $\psi \in \textsc{Form}(X)$, the formula $\overline{\psi}$ is obtained from ψ by removing its *fixed variables* (see above) using a SAT solver. The root of $\mathit{Stree}(\overline{\varphi})$ is labeled by acc, and the roots of $\mathit{Stree}(\overline{\psi_1(\boldsymbol{p})}), \ldots, \mathit{Stree}(\overline{\psi_n(\boldsymbol{p})})$ are labeled by out. The graph B_{φ,e_2} is constructed in the same way, again using the labels acc and out.

Theorem 13. *If B_{φ,e_1}, B_{φ,e_2} are isomorphic, then $e_1 \sim_\varphi e_2$.*

The procedure $\textsc{Experiments}(\varphi)$ is used to compute decision trees for ranking strategies and optimal worst/average case strategies in the following way. Let $\tau[r, \preceq]$ be a ranking strategy such that for all $e_1, e_2 \in E$ and $\varphi \in \textsc{Know}$ we have that $e_1 \sim_\varphi e_2$ implies $r(e_1) = r(e_2)$. Note that all ranking functions introduced in this section satisfy this property. The decision tree $\mathit{Tree}_{\tau[r,\preceq]}$ is computed top-down. When we need to determine the label of a given node u where the associated accumulated knowledge is Ψ_u, we first check whether $|\mathit{Val}(\Psi_u)| = 1$ using a SAT solver. If it is the case, we label u with the only valuation of $\mathit{Val}(\Psi_u)$. Otherwise, we need to compute the experiment $\tau[r, \preceq](\Psi_u)$. It follows immediately that $\tau[r, \preceq](\Psi_u)$ is contained in $S_{\Psi_u} := \textsc{Experiments}(\Psi_u)$. Hence, we label u with the least element of $\{e \in S_{\Psi_u} \mid \mathit{Updates}[\Psi_u, e] = \mathit{Min}\}$ w.r.t. $\preceq$, where $\mathit{Min} = \min\{\mathit{Updates}[\Psi_u, e'] \mid e' \in S_{\Psi_u}\}$. This element is computed with the help of a SAT solver.

The way of computing a decision tree for an optimal worst/average case strategy is more involved. Let $\textsc{Wopt}_\mathcal{G}$ and $\textsc{Aopt}_\mathcal{G}$ be the sets of all knowledge-based strategies which are worst case optimal and average case optimal, respectively. First, observe

```
1  Function Optimal(φ, upper)
2      if |Val(φ)| = 1 then return ⟨v, 0⟩ where v ∈ Val(φ)
3      if φ is cached then return  the cached result
4      [W] if ⌈log_Out(|Val(φ)|)⌉ > upper then return ⟨err, ∞⟩
5      S_φ := Experiments(φ)
6      best := upper;  e_φ := some element of S_φ
7      for e ∈ S_φ do
8          val := 0
9          for ψ ∈ Φ(e) do
10             if SAT(φ ∧ ψ) then
11                 ⟨e_ψ, C_ψ⟩ := Optimal(φ ∧ ψ, best − 1)
12                 [W] val := max(val, 1 + C_ψ)
13                 [A] val := val + |Val(φ ∧ ψ)| · (1 + C_ψ)
14         [A] val := val / |Val(φ)|
15         if val ≤ best then best := val; e_φ := e
16     Cache the result ⟨e_φ, best⟩ for φ
17     return ⟨e_φ, best⟩
```

Fig. 2. Computing optimal strategies.

that if $\tau \in \text{WOPT}_{\mathcal{G}}$ and $\tau(\varphi) = e$ for some $\varphi \in \text{KNOW}$, then for every $e' \in E$ where $e \sim_\varphi e'$ there is $\tau' \in \text{WOPT}_{\mathcal{G}}$ such that $\tau'(\varphi) = e'$. Hence, we can safely restrict the range of $\tau(\varphi)$ to $\text{EXPERIMENTS}(\varphi)$. Further, if $\tau(\varphi) = e$ and $\varphi' \equiv \pi(\varphi)$ for some $\pi \in \Pi$, we can safely put $\tau(\varphi') = \pi(e)$. The same properties hold also for the strategies of $\text{AOPT}_{\mathcal{G}}$.

A recursive function for computing a worst/average case optimal strategy is shown in Fig. 2. The function is parameterized by $\varphi \in \text{KNOW}$ and an upper bound on the worst/average number of experiments performed by an optimal strategy for the initial knowledge φ. The function returns a pair $\langle e_\varphi, C_\varphi \rangle$ where e_φ is the experiment selected for φ and C_φ is the worst/average number of experiments that are needed to solve the game for the initial knowledge φ. Hence, the algorithm is invoked by $\text{OPTIMAL}(\varphi_0, \infty)$. Note that the algorithm caches the computed results and when it encounters that φ is π-symmetric to some previously processed formula, it uses the cached results immediately (line 3). The lines executed only when constructing the worst (or average) case optimal strategy are prefixed by $[W]$ (or $[A]$, respectively). At line 4, the constant Out is equal to $\max_{(k,P,\Phi)\in T} |\Phi(t)|$. Obviously, we need at least $\lceil \log_{Out}(|\,Val(\varphi)|) \rceil$ experiments to distinguish among the remaining $|\,Val(\varphi)|$ alternatives.

3 Cobra: The Tool and Experimental Results

COBRA [1] is a command-line tool envoked as follows:

```
cobra [-m <mode>] [-s <sat solver>] [other options] <file>
```

The `<file>` contains a deductive game description (the syntax implements Definition 1). The `<mode>` can be either `overview`, `analysis`, `optimal-worst`, or `optimal-average`. The `overview` mode serves for basic consistency checks (in particular, the *well-formed* condition is verified, see Section 2). The `analysis` mode allows

	CCP 26 ($\approx 10^{26}$ exp.)		**CCP 39** ($\approx 10^{46}$ exp.)		**CCP 50** ($\approx 10^{64}$ exp.)	
Exp.No.	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
hline 1	13.0	13.0	19.0	19.0	25.0	25.0
2	4,365.0	861.7	26,638.7	3,318.0	83,625.0	8,591.0
3	603.0	36.4	2,263.0	88.1	5,733.4	172.2
4	76.3	4.2	214.7	7.2	405.1	10.4
5	-	-	-	-	153.2	4.1

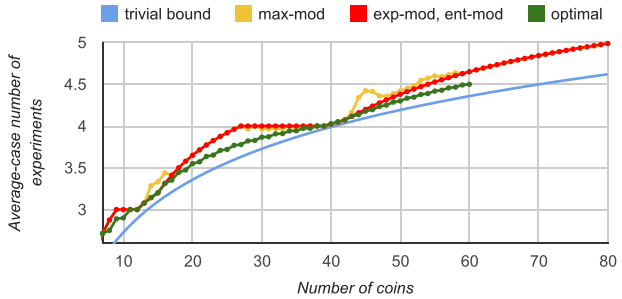
	MM 3x8 (512 exp.)				**MM 4x6** (1296 exp.)				**MM 5x3** (243 exp.)			
	max-models		parts		max-models		parts		max-models		parts	
Exp.No.	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
1	5.00	3.00	5.00	3.00	15.00	5.00	15.00	5.00	41.00	5.00	41.00	5.00
2	70.13	17.38	70.13	17.38	144.82	34.91	337.23	106.62	243.00	59.25	243.00	59.25
3	144.50	72.31	147.29	87.83	587.54	243.40	819.49	580.03	243.00	121.45	243.00	186.90
4	134.25	71.54	155.14	87.98	791.30	344.02	819.68	417.02	-	-	-	-
5	91.36	25.36	100.46	31.97	-	-	334.33	95.83	-	-	-	-

Table 2. The size of S^1_φ and S_φ for selected deductive games.

to analyze the worst/average case complexity of ranking strategies for several ranking functions. Currently, Cobra supports `max-models`, `exp-models`, `ent-models`, `part`, `min-fixed`, and `exp-fixed` ranking functions, where the first four functions minimize the worst-case number of remaining secret codes, the expected number of remaining secret codes, the entropy of the numbers of remaining secret codes, and the number of satisfiable outcomes, respectively, and the last two functions maximize the (expected) number of fixed variables (precise definitions can be found in Appendix **??**). Finally, the `optimal-worst` and `optimal-average` are the modes where Cobra computes the worst and the average case optimal strategies, respectively. The optional `-s` switch allows to specify the SAT solver used by Cobra for evaluating the supported ranking functions (currently available options are MiniSat [9] and PicoSAT [5]). Cobra also uses the tool Bliss [15] for checking graph isomorphism to determine equivalent experiments. The source code, installation instructions, examples, and a more detailed specification of Cobra's functionality are available freely at GitHub [1].

In the rest of this section we briefly describe some experimental results achieved with Cobra. In the first part, we demonstrate the efficiency of the algorithm for eliminating symmetric experiments discussed at the end of Section 2. In the second part, we show that Cobra is powerful enough to produce new results about existing deductive games and their variants.

The functionality of Experiments(φ) can be well demonstrated on CCP and Mastermind. Consider CCP with 26, 39, and 50 coins. Table 2 (top) shows the *average* size of S^1_φ and S_φ when computing the i-th experiment in the decision tree for the `max-models` ranking strategy. The total number of experiments for 26, 39 and 50 coins is larger than 10^{26}, 10^{46}, and 10^{64}, respectively. Observe that for 26 and 39 coins, only four experiments are needed to reveal the counterfeit coin, and hence the last row is empty. Note that in the first round, *all* equivalent experiments are discovered already in the first phase, i.e., when computing S_1. These experiments correspond to the number of coins that can be weighted (e.g., for 50 coins we can weight $1{+}1, \ldots, 25{+}25$ coins, which gives 25 experiments). In the second round, when we run Experiments(φ) for three different formulae $\varphi \in$ Know, the average size of S^1_φ is already larger, and the second phase (eliminating equivalent experiments) further reduces the average size of the resulting S_φ.



Average-case			
Size	MM	MM+col	MM+pos
2x8	3.67187	3.64062	2
3x6	3.19444	3.18981	3
4x4	2.78516	2.74609	2.78516
Worst-case			
Size	MM	MM+col	MM+pos
2x8	5	5	2
3x6	4	4	3
4x4	3	3	3

Table 3. The average/worst case complexity of selected deductive games.

A similar table for Mastermind is shown in Table 2 (bottom). Here we consider three variants with 3/8, 4/6, and 5/3 pegs/colors. The table shows the average size of S_φ when computing the i-th experiment in the decision trees for `max-models` and `parts` ranking strategies. Note that for Mastermind, the reduction is more efficient for more colors and less pegs, and that the values for the two ranking strategies significantly differ, which means that they divide the solution space in a rather different way.

Now we present examples of results obtained by running our tool that, to the best of our knowledge, have not yet been published in the existing literature about deductive games. Our first example concerns CCP. While the worst case complexity of CCP is fully understood [8], we are not aware of any results about the *average* case complexity of CPP. Using Cobra, we were able to compute the *average-case optimal strategy* for up to 60 coins. Further, we can compare the average-case complexity of an optimal strategy with the average-case complexities of various ranking strategies, which can be synthesized for even higher number of coins (more than 80). The results are summarized in the graph of Table 3 (left). The precise values shown in the plot can be found in [16].

As the last example, we consider two variants of Mastermind: MM+col, where we can also ask for all pegs colored by a given color, and MM+pos, where we can also ask for the color of a given peg. These extensions are inspired by the API-level attacks mentioned in Section 1. Using Cobra, we can compute the optimal worst/average case complexity for 2/8, 3/6, and 4/4 pegs/colors. The results are summarized in Table 3 (right). When comparing these results to "classical" results about Mastermind, the following subtle difference in game rules must be taken into account: Plays of "our" deductive games terminate as soon as we obtain enough information to reveal the secret code. The "classical" Mastermind terminates when the secret code is "played", which may require an extra experiment even if the code is already known. Our numbers are valid for the first setup.

4 Conclusions

The results produced by Cobra witness that non-trivial deductive games can be solved by a generic tool. The main advantage of Cobra is its *versatility*; small changes in the structure of the secret code and/or experiments can easily be reflected in the input description, which greatly simplifies the analysis of new versions of

security protocols, new forms of attacks, etc. The challenge is to push the frontiers of fully automatic analysis of deductive games even further. Obviously, there are many ways of improving the functionality of Cobra by elaborating the concepts presented in this paper. The interface to SAT solvers can also be tuned, there is a lot of space for parallelism, etc. One may also try alternative approaches to modeling and solving deductive games based on constraint solving or artificial intelligence techniques.

References

1. COBRA, the COde-BReaking game Analyzer. https://github.com/myreg/cobra (2014)
2. Bento, L., Pereira, L., Rosa, A.: Mastermind by evolutionary algorithms. In: Proceedings of the International Symposium on Applied Computing. pp. 307–311. ACM (1999)
3. Berghman, L., Goossens, D., Leus, R.: Efficient solutions for Mastermind using genetic algorithms. Computers & Operations Research 36(6), 1880–1885 (2009)
4. Bernier, J., Herraiz, C., Merelo, J., Olmeda, S., Prieto, A.: Solving Mastermind using gas and simulated annealing: A case of dynamic constraint optimization. In: Parallel Problem Solving from Nature - PPSN IV, International Conference on Evolutionary Computation. The 4th International Conference on Parallel Problem Solving from Nature. Lecture Notes in Computer Science, vol. 1141, pp. 554–563. Springer (1996)
5. Biere, A.: PicoSAT essentials. Journal on Satisfiability, Boolean Modeling and Computation 4(2–4), 75–97 (2008)
6. Bond, M., Zieliński, P.: Decimalisation table attacks for PIN cracking. Tech. Rep. UCAM-CL-TR-560 arXiv:1407.3926, University of Cambridge (2003)
7. Chen, S.T., Lin, S.S., Huang, L.T., Hsu, S.H.: Strategy optimization for deductive games. European Journal of Operational Research 183, 757–766 (2007)
8. Dyson, F.: The problem of the pennies. The Mathematical Gazette 30, 231–234 (1946)
9. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proceedings of SAT 2003. Lecture Notes in Computer Science, vol. 2919, pp. 502–518. Springer (2004)
10. Erdös, P., Rényi, A.: On two problems of information theory. Magyar Tud. Akad. Mat. Kutató Int. Közl 8, 229–243 (1963)
11. Gagneur, J., Elze, M., Tresch, A.: Selective phenotyping, entropy reduction, and the Mastermind game. BMC Bioinformatics 12(406) (2011)
12. Goodrich, M.: The Mastermind attack on genomic data. In: Proceedings of 30th IEEE Symposium on Security and Privacy. pp. 204–218. IEEE (2009)
13. Guy, R., Nowakowski, R.: Coin-weighting problems. The American Mathematical Monthly 102(2), 164–167 (1995)
14. Irving, R.: Towards an optimum Mastermind strategy. Journal of Recreational Mathematics 11(2), 81–87 (1978–79)
15. Junttila, T., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX 2007). pp. 135–149. SIAM (2007)
16. Klimoš, M., Kučera, A.: Strategy synthesis for general deductive games based on SAT solving. CoRR abs/1407.3926 (2015)
17. Knuth, D.: The computer as Mastermind. Journal of Recreational Mathematics 9(1), 1–6 (1976)
18. Kooi, B.: Yet another Mastermind strategy. ICGA Journal 28(1), 13–20 (2005)
19. Koyama, K., Lai, T.: An optimal Mastermind strategy. Journal of Recreational Mathematics 25(4), 251–256 (1993)
20. Neuwirth, E.: Some strategies for Mastermind. Zeitschrift für Operations Research 26, 257–278 (1982)
21. Steel, G.: Formal analysis of PIN block attacks. Theoretical Computer Science 367(1–2), 257–270 (2006)