Masaryk University
Faculty of Informatics

Algorithmic Analysis of Code Breaking Games

Master's Thesis

Miroslav Klimoš

Brno, 2014

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Advisor: prof. RNDr. Antonín Kučera, Ph.D.

Keywords

code braking games,
searching games,
counterfeit coin,
mastermind,
sat solving,
greedy strategy

Contents

1 Introduction

2 Code Breaking Games

We introduce a few examples of Code Breaking Games in this Chapter. The counterfeit coin problem and Mastermind game are quite well known, the other examples are based on various board games or less known logic puzzles. We briefly summarize related research for each game, give a list of references and discuss its application.

Our goal in this work is neither to answer the research questions nor to study possible generalizations. Out goal is to create a general formalism and a computer language which could be used to describe arbitraty Code Breaking Game, if possible.

2.1 The Counterfeit Coin

The problem of finding a counterfeit coin among regular coins in the fewest number of weighings on a pair of scales balance is the folklore of recreational mathematics.

In all problems of this kind, you can use the scales only to weight the coins. You put some coins on the left pan, the same number of coins on the right pan and get one of the 3 possible outcomes. Either both the sides weight the same (denoted "=") or the left side is lighter ("<"), or the right side is lighter (">").

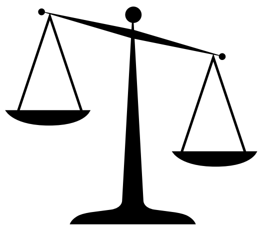

Figure 2.1: Balance scales (illustrative image)[1].

The standard, easiest variant can be formulated as follows.

Problem 1 (The nine coin problem). *You are given* $n \geq 3$ *(typically 9) coins, all expect one have the same weight. The counterfeit coin is known to be lighter. Determine the coin in the minimal number of weightings.*

This problem is very easy as one can use *ternary search* algorithm. In short, we divide the coins into thirds, put one third agains another on scales. If both sides weight the same, the counterfeit coin must be in the last third, otherwise it must be in the lighter third. In this way, the size of search space reduces by a factor of 3 in one weighting, which is clearly optimal.

In 1940s, a more complicated variant was introduced by Grossman[1].

Problem 2 (The Twelve Coin Problem). *You are given* $n \geq 3$ *(typically 12) coins, exactly one of which is counterfeit, but it is not known if it is heavier or ligher. Determine the unique coin and its weight relavite to others in the minimal number of weightings.*

1. Adapted from `http://pixabay.com/en/justice-silhouette-scales-law-147214`, under CC0 1.0 Licence.

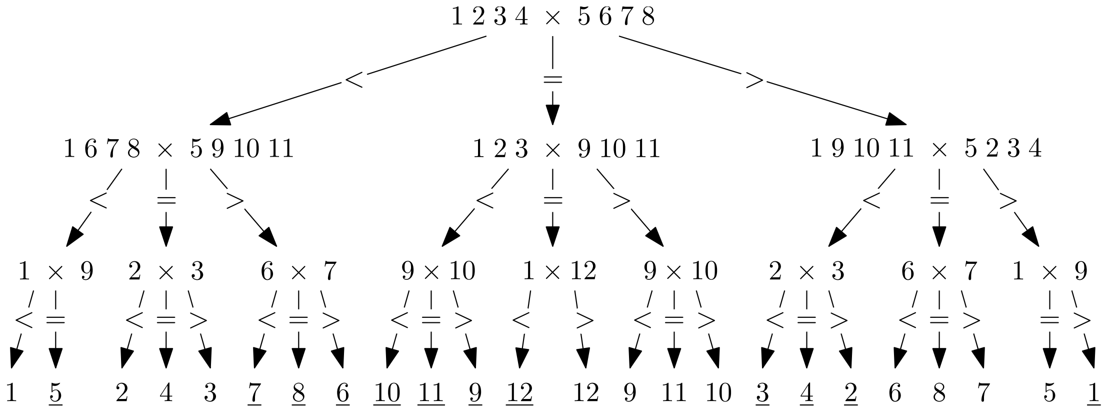

Figure 2.2: Decision tree for The Twelve Coin Problem.
Leaf x means that the coin number x is lighter, $\underline{x}$ means that the coin number x is heavier.

The optimal solution for $n = 12$ requires 3 weightings and one of the optimal strategies is shown in Figure 2.2 as a decision tree.

The research usually focuses on bounds on the maximal value of n for which the problem can be solved in w weighings, for a given w. Thus a solution of a problem is usually formulated as a theorem like the following one.

Theorem 3 (Dyson, [2]). *There exists a scheme that determines the counterfeit coin and its type in Problem 2 with w weighings, if and only if*

$$3 \leq n \leq \frac{3^w - 3}{2}.$$

Proof. We show the main part of the original Dyson's proof[2] here because of its elegant combinatorial idea. We show a scheme for $n = \frac{1}{2}(3^w - 3)$.

Let us number the coins from 1 to n. To a coin number i, we assing two labels from $\{0,1,2\}^w$ – those corresponding the numbers i and $3^w - 1 - i$ in ternary form. Notice that all possible labels are used exactly once, except for $0^w, 1^w$ and 2^w, which were not assigned to any coin. The labeling has the property that you can get one label of a coin from the other by substituing 0 by 2 and 2 by 0.

A label is called "clockwise" if the first change of digit in it is the change from 0 to 1, from 1 to 2, or from 2 to 0. Otherwise, it is called "anticlockwise". Thanks to the property we mentioned, one of the labels of a coin is always clockwise and the other is anticlockwise.

Let $C(i,d)$ be a set of coins such that i-th symbol in its clockwise label is d. Since a permutation changing 0 to 1, 1 to 2 and 2 to 0 transfers coins from $C(i,0)$ to $C(i,1)$, from $C(i,1)$ to $C(i,2)$ and from $C(i,2)$ to $C(i,0)$, all the sets $C(i,d)$

coin	label 1	label 2
1	**001**	221
2	002	**220**
3	**010**	212
4	**011**	211
5	**012**	210
6	020	**202**
7	021	**201**
8	022	**200**
9	100	**122**
10	101	**121**
11	102	**120**
12	110	**112**

Experiments:

1) 1, 3, 4, 5 × 2, 6, 7, 8
2) 1, 6, 7, 8 × 2, 9, 10, 11
3) 2, 3, 8, 11 × 5, 6, 9, 12

Solution:

the coin labelled $a_1a_2a_3$, where a_i is the outcome of i-th experiment.

Figure 2.3: Demonstration of the ternary label construction for $n = 12$.

contain exactly $n/3$ coins. Now, let i-th experiment be the weighing of the coins $C(i,0)$ agains $C(i,2)$. It remains to show that the experiments uniquely determine the counterfeit coin. Let a_i be 0, 1, or 2 if the result of i-th experiment is left side is lighter, both are the same, or right side is lighter, respectively.

If the counterfeit code is overweight, the i-th symbol of its clockwise label must be a_i. On the other hand, if it is underweight, the i-th symbol of its anticlockwise label must be a_i. The solution of the problem is therefore the coin with the label $a_1a_2\dots a_w$ and is heavier than others if and only if this label is clockwise. Figure 2.3 shows an example of the construction for $n = 12 = \frac{1}{2}(3^3 - 3)$.

The case $n < \frac{1}{2}(3^w - 3)$ can be done similarly with some modifications to the labelling. However, the scheme makes use of a genuine coin that was discovered in the first weighing and, therefore the following experiments depend on the outcome of the first. Finally, the prove that the coin cannot be detected if $n > \frac{1}{2}(3^w - 3)$ can be done using information theory.

Naturally, the problem was generalized in various ways and studied by many authors. In “Coin-Weighing Problems”[3], Guy and Nowakovski gave a great overview of the research in the area until 1990s with an extensive list of references. We list the most interesting variants and generalizations below.

- **Weight of counterfeit coin.** Either it is known whether the counterfeit coin is lighter or heavier, or it is not. The first one allows for more generalizations due to its simpler nature but both problems have been heavily researched.
- **Number of counterfeit coins.** In the most common case, there is exactly one counterfeit coin, which allows for natural generalizations. First, a variant of Problem 1 with 2 or 3 counterfeit coins was studied[4][5], then with m

counterfeit coins in general[6]. Some authors studied the problem for unknown number of counterfeit coins [7], or for *at most m* counterfeit coins[8].

- **Additional regular coin(s).** In some cases, it may help if you are given an additional coin (or more coins), which is guaranteed not to be counterfeit. For example, for $n = 13$ in Problem 2, you need 4 weightings. However, if you are given this one extra coin, you can determine the solve in just 3 weightings[2].
- **Non-adaptive strategies.** In this popular variant of the problem you have to announce all experiments in advance and then just collect the result. In other words, later weighings must not depend on the outcomes of the earlier weighings. Notice that the scheme constructed in the proof of Theorem 3 for $n = \frac{1}{2}(3^w - w)$ is indeed non-adaptive. However, the original proof uses an adaptive scheme for a smaller n. This was later fixed, showing that there always exists an optimal scheme for Problem 2 which is non-adaptive[9].
- **Unreliable balance.** This generalization introduces the possibility that one (or more) answers may be erroneous. The problem of errors/lies in general search problems is well studied, see [10]. It was applied on the counterfeit coin problem (Problem 1 variant) in [11] with at most one erroneous outcome or in [11] with two.
- **Multi-arm balance.** In this variant, your balance has k arms. You put the same number of coins on every arm and you get either the information that all weight the same or which arm is lighter or heavier than others[12].
- **Parallel weighting.** In this generalization, you have 2 (or k, in general) balance scales, you can weight different coins on the two scales simultaneously and it counts as only one experiment[13]. The motivation here is that the weighing takes some not insignificant time, you have more scales and strive to minimize the time the whole process takes.

2.2 Mastermind

Mastermind is a classical code-breaking board game for 2 players, invented by Mordecai Meirowitz in 1970. One player has the role of a *codemaker* and the other of a *codebreaker*. First, the codemaker chooses a secret code of n pegs of c colors. Then a codebreaker tries to reveal the code by making guesses. The codebreaker evaluates the guesses using black and white markers. Black markers correspond to positions at which the code and the guess matches, white marker correspond to a color peg which is both in the code and the guess, but at different positions. The markers in the answer are not ordered, so the codebreaker does not know, which marker correspond to which peg in the guess. The codebreaker strives to minimize the number of guesses.

More formally, let C be a set of colors of size c. Define a distance $d : C^n \times C^n \to \mathbb{N}_0 \times \mathbb{N}_0$ of two color sequences by $d(u,v) = (b,w)$, where

$$b = |\{i \in \mathbb{N} \mid u[i] = v[i]\}|$$
$$w = \sum_{j \in C} \min\left(\left|\{i \mid u[i] = j\}\right|, \left|\{i \mid v[i] = j\}\right|\right) - b.$$

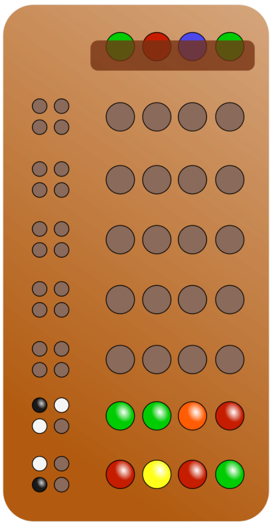

Figure 2.4: Mastermind game (illustrative image)[2].

If the codemaker's secret code is h and the codebreaker's guess is g, the guess should be evaluated with b black pegs and w white pegs, where $(b,w) = d(h,g)$. Therefore, if the codebreaker have guessed $g_1, g_2, \ldots, g_k$ and the results were $(b_1,w_1), \ldots, (b_k,w_k)$, the search space is reduced to codes

$$\{u \in C^n \mid \forall i \le k.\ d(u,g_i) = (b_i,w_i)\}.$$

Another way of looking at the guess evaluation is using *maximal matching* of the pegs in the code h and the guess g. A matching is a set of pair-wise non-adjacent edges between pegs in the code (represented by $(0,i)$ for $1 \le i \le n$) and pegs in the guess (represented by $(1,i)$ for $1 \le i \le n$). Let M be a maximal matching such that

1. an edge connects only pegs of the same color, i.e. if $((0,i),(1,i)) \in M$, then $h[i] = g[j]$, and
2. if $h[i] = g[i]$ then $((0,i),(1,i)) \in M$.

Maximal means that no edge can be added without breaking one of the conditions. The edges in M correspond to the markers in the response, a marker being black if and only if the corresponding edge connects $(0,i)$ with $(1,i)$ for some i.

Variants and generalizations

The old game of *Bulls and Cows* has very similar principle to Mastermind, it only uses letters instead of colors and does not allow repetitions.

Static Mastermind is a variant of the game in which all guesses must be made at one go. The codebreaker prepares a set of guesses, then the codemakers evaluates each one as usual and the codebreaker must determine the code from the outcomes. This corresponds to so-called *non-adaptive* strategies for The Counterfeit Coin Problem.

2. Adapted from `http://commons.wikimedia.org/wiki/File:Mastermind_beispiel.svg`, by Thomas Steiner under GFDL.

Known results and related research

one-step look-ahead heuristics, tabulka...

2.3 Other Problems and Applications

Black Box

Bags of Gold

Code 777

String matching

Generalized Mastermind

3 General model

3.1 Notation and Terminology

Let $\texttt{Form}_X$ be the set of all prepositional formulas over the set of variables X; $\texttt{Val}_X$ be the set of all valuations (boolean interpretation) of variables X. Formulas $\varphi_0, \varphi_1 \in \texttt{Form}_X$ are (semantically) equivalent, written $\varphi_0 \equiv \varphi_1$, if $v(\varphi_0) = v(\varphi_1)$ for all $v \in \texttt{Val}_X$. We say that v *is a model of* φ or that v *satisfies* φ if $v(\varphi) = 1$.
For a formula $\varphi \in \texttt{Form}_X$, let $\#_X\varphi = |\{v \in \texttt{Val}_X \mid v(\varphi) = 1\}|$ be the number of models of φ (valuations satisfying φ). We often omit the index X if it is clear from the context.
The set of all permutations of a set X (bijections $X \to X$) is denoted by $\texttt{Perm}_X$ and $\texttt{id}_X$ is the identity permutation.

3.2 Formal definition

In this section, we formally define Code Breaking Games within the framework of propositional logic, where we represent the secret code as a valuation of propositional variables. The game is represented as a *set of variables*, *initial restriction* (a formula that is guaranteed to be satisfied), and a set of *possible experiments*. A finite set of possible *outcomes* is associated with each experiment. Outcome is a propositional formula that represents the partial information, which the codebreaker can gain from the experiment.
The number of experiments is typically very large (such as 36894 for the Counterfeit-coin Problem **??**) but most of them have same structure and yield similar outcomes. Therefore we opt for a compact representation of an experiment as a pair (type of experiment, parametrization), where parametrization is a string over a defined alphabet. This whole idea is formalized below.

Definition 4 (Code Breaking Game). A *Code Breaking Game* is a septuple $\mathcal{G} = (X, \varphi_0, T, \Sigma, E, F, \Phi)$, where

- X is a finite set of propositional variables,
- $\varphi_0 \in \texttt{Form}_X$ is a satisfiable propositional formula,
- T is a finite set of types of experiments,
- Σ is a finite alphabet,
- $E \subseteq T \times \Sigma^\star$ is an *experiment* relation, and
- F is a finite collection of functions of type $\Sigma \to X$,
- $\Phi : T \to 2^{\texttt{PForm}_{X,F,\Sigma}}$ is an *outcome function* such that $\Phi(t)$ is finite for any $t \in T$. Definition of $\texttt{PForm}$ follows (Definition 5).

Definition 5 (Parametrized formula). A set of *parametrized formulas* $\texttt{PForm}_{X,F,\Sigma}$ is a set of all strings ψ generated by the following grammar:

$$\psi ::= x \mid f(\$n) \mid \psi \circ \psi \mid \neg\psi,$$

where $x \in X$, $f \in F$, $n \in \mathbb{N}$, and $\circ \in \{\wedge, \vee, \Rightarrow\}$. By $\psi(p)$ we denote application of a parametrization $p \in \Sigma^\star$ on a formula ψ, which is defined recursively on the structure of ψ in the following way:

$$\begin{aligned}(x)(p) &= x,\\ (f(\$n))(p) &= f(p[n]),\\ (\psi_1 \circ \psi_2)(p) &= \psi_1(p) \circ \psi_2(p),\\ (\neg\psi)(p) &= \neg(\psi(p)).\end{aligned}$$

We use the special symbol \$ in $f(\$n)$ so that n cannot be mistaken for the argument of f, which is n-th symbol of the parametrization. Note that if $f(\$n)$ appears in ψ and $|p| < n$, then $\psi(p)$ is undefined.
For the sake of simplicity, let us denote the set of possible outcomes for an experiment $e = (t, p) \in E$ by $\Phi(e) = \{\psi(p) \mid \psi \in \Phi(t)\}$.
The compact representation with parametrized formulas does not restrict the class of games that can fit this definition. If no two experiments can be united under the same type, every experiment can have its own type and allow only one possible parametrization.

Definition 6 (Solving process). An *evaluated experiment* is a pair (e, φ) such that $\varphi \in \Phi(e)$. Let us denote the set of evaluated experiments by Ω.
A *solving process* is a finite or infinite sequence of evaluated experiments.

For simplicity, we omit the brackets around the pairs and write

$$\lambda = e_1, \varphi_1, e_2, \varphi_2, \ldots$$

Let

- $|\lambda|$ denote the length of the sequence,
- $\lambda(k) = e_k$ denote the k-th experiment,
- $\lambda[k] = \varphi_k$ denote the k-th outcome,
- $\lambda[1..k] = e_1, \varphi_1, \ldots, e_k, \varphi_k$ denote the prefix of length k, and

- $\lambda\langle k\rangle = \varphi_0 \wedge \varphi_1 \wedge \ldots \wedge \varphi_k$ denote the accrued knowledge after the first k experiments (including the initial restriction φ_0). For finite λ, let $\lambda\langle\rangle = \lambda\langle|\lambda|\rangle$ be the overall accrued knowledge.

We denote by $\mathtt{Val}^* = \{v \in \mathtt{Val}_X \mid v(\varphi_0) = 1\}$ the set of valuations that satisfy φ_0 and by $\mathtt{Form}^* = \{\lambda\langle\rangle \mid \lambda \in \Omega^*\}$ the set of *reachable formulas*.

Let us now describe the course of the game in the defined terms. First, the codemaker choose a valuation v from $\mathtt{Val}^*$. Second, the codebreaker chooses a type $t \in T$ and a parametrization $p \in \Sigma^*$ such that $(t, p) \in E$. Third, the codemaker gives the codebreaker a formula $\varphi \in \Phi((t, p))$, which is satisfied by the valuation v. Then the evaluated experiment $((t, p), \varphi)$ is appended to the (initially empty) solving process λ and they continue with the second step. The game continues until $\#\lambda\langle\rangle = 1$, which corresponds to the situation in which the codebreaker can uniquely determine the code.

So that the codemaker can always fulfill the third step, there must be a formula $\varphi \in \Phi(e)$ satisfied by any valuation. Although it might make sense to allow multiple satisfied formulas, we restrict ourselves to games where the outcome is uniquely defined for given valuation.

Definition 7 (Well-formed game). A code-breaking game is *well-formed* if for all $e \in E$,

$$\forall v \in \mathtt{Val}^* .\ \exists \text{ exactly one } \varphi \in \Phi(e)\ .\ v(\varphi) = 1$$

As the semantics of non-well-formed games is unclear, we focus only on well-formed games and, be default, we suppose a game to be well-formed if not stated otherwise.

Example 8 (Fake-coin problem). Fake-coin problem with n coins, one of which is fake, can be formalized as a code breaking game $\mathcal{F}_n = (X, \varphi_0, T, \Sigma, E, F, \Phi)$.

- $X = \{x_1, x_2, \ldots, x_n, y\}$,
  $\varphi_0 = \mathtt{Exactly}_1\ \{x_1, \ldots, x_n\}$.
  Intuitively, variable x_i tells weather the coin i is fake. Variable y tells weather it is lighter or heavier. Formula φ_0 says that exactly one coin is fake.
- $T = \{w_2, w_4, \ldots, w_n\}$,
  $\Sigma = \{1, 2, \ldots, n\}$,
  $E = \bigcup_{1 \le m \le n/2} \{(w_{2m}, p) \mid p \in \{1, \ldots, n\}^{2m}, \forall x \in X. \#_x(p) \le 1\}$.
  There are $n/2$ types of experiment – according to the number of coins we put on the weights. The alphabet contains natural numbers up to n and possible parametrizations for w_{2m} are strings of length $2m$ with no repetitions.
- $F = \{f_x\}$, where $f_x(i) = x_i$ for $1 \le i \le n$,
  $\Phi(w_m) =$

$$\{((f_x(\$1) \vee \ldots \vee f_x(\$m)) \wedge \neg y) \vee ((f_x(\$m+1) \vee \ldots \vee f_x(\$2m)) \wedge y),$$
$$((f_x(\$1) \vee \ldots \vee f_x(\$m)) \wedge y) \vee ((f_x(\$m+1) \vee \ldots \vee f_x(\$2m)) \wedge \neg y),$$
$$\neg(f_x(\$1) \vee \ldots \vee f_x(\$2m))\}.$$

There are 3 possible outcomes of every experiment. First, the right side is heavier. This happens if the fake coin is lighter and it appears in the first half of the parametrization, or if it is heavier and it appears in the second half. Second, analogously, the left side is heavier. Third, the weights are balanced if the fake coin do not participate in the experiment.

Example 9 (Fake-coin problem, alternative). For demonstration purposes, here is another formalization of the same problem. $\mathcal{F}'_n = (X, \varphi_0, T, \Sigma, E, F, \Phi)$.

- $X = \{x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n\}$,
  $\varphi_0 = \texttt{Exactly}_1 \{x_1, \ldots, x_n, y_1, \ldots, y_n\}$.
  Variable x_i tells that the coin i is lighter, variable y_i tells that the coin i is heavier. Formule φ_0 says that exactly one coin is different.
- T, Σ, E is defined as in Example 8.
- $F = \{f_x, f_y\}$, where $f_x(i) = x_i$ and $f_y(i) = y_i$ for $1 \leq i \leq n$,

$$\begin{aligned}\Phi(w_m) = \{&(f_x(\$1) \vee \ldots \vee f_x(\$m)) \vee (f_y(\$m+1) \vee \ldots \vee f_y(\$2m)),\\ &(f_y(\$1) \vee \ldots \vee f_y(\$m)) \vee (f_x(\$m+1) \vee \ldots \vee f_x(\$2m)),\\ &\neg(f_x(\$1) \vee \ldots \vee f_x(\$2m) \vee f_y(\$1) \vee \ldots \vee f_y(\$2m))\}.\end{aligned}$$

Example 10 (Mastermind). Mastermind puzzle with n pegs and m colors can be formalized as a code breaking game $\mathcal{M}_{n,m} = (X, \varphi_0, T, \Sigma, E, F, \Phi)$.

- $X = \{x_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m\}$,
  $\varphi_0 = \bigwedge \{\texttt{Exactly}_1 \{x_{i,j} \mid 1 \leq j \leq m\} \mid 1 \leq i \leq n\}$.
  Variable $x_{i,j}$ tells whether there is the color j at position i. Formula φ_0 says that there is exactly one color at each position.
- $T = \{g\}$,
  $\Sigma = C$,
  $E = \{(g, p) \mid p \in \Sigma^n\}$.
  There is only one type of experiment, parametrization of which is any sequence of colors of length n.
- $F = \{f_1, \ldots, f_n\}$, where $f_i(c) = x_{i,c}$ for $1 \leq i \leq n$,
  $\Phi(g) = \{\text{Outcome}(b, w) \mid 0 \leq b \leq n, 0 \leq w \leq n, b + w \leq n\}$, where Outcome function is computed by the algorithm described below.

As described in the introduction of the Mastermind problem, the outcome corresponds to some maximal matching between the pegs in the code and in the guess. The idea here is to generate all matchings corresponding to a given outcome, generate a formula that expresses validity of the matching for a given experiment and put them into a big disjunction.

The computation of Outcome (b, w) works as follows. First, we generate all the matchings. Let $P = \{1, 2, ..n\}$ be the set of positions.

- Select $B \subseteq P$ such that $|B| = b$. These are the positions at which the color in the code and in the guess matches and they correspond to the black markers.
- Select $W \subseteq P \times P$ such that $|W| = w$, $p_1(W) \cap B = \emptyset$, and $p_2(W) \cap B = \emptyset$, where p_1, p_2 are the projections. These correspond to the white markers – $(i, j) \in W$ means that the color at position i in the guess is at position j in the code.

Next, for each combination (B, W), we generate a conjunction in the following way:

- For $i \in B$, we add $f_i(\$i)$.
- For $(i, j) \in W$, we add $\neg f_i(\$i) \wedge f_j(\$i)$.
- For $(i, j) \in (P \setminus B \setminus p_1(W)) \times (P \setminus B \setminus p_2(W))$, we add $\neg f_j(\$i)$. This guarantees that the matching is maximal.

The result is a disjunction of all these clauses, which effectively enumerates all the cases. For example, for $n = 4$ the result of Outcome$(1, 1)$ starts with

$$(\neg f_0(\$0) \wedge \neg f_1(\$1) \wedge \neg f_1(\$2) \wedge \neg f_2(\$1) \wedge \neg f_2(\$2) \wedge f_3(\$3)) \vee (\neg f_0(\$0) \wedge \neg f_0(\$1) \wedge \neg f_0(\$2) \wedge \neg f_1(\$0) \wedge \neg f_1(\$1) \wedge \neg f_2(\$1) \wedge \neg f_2(\$2) \wedge f_3(\$3)) \vee (\neg f_0(\$0) \wedge \neg f_0(\$1) \wedge \neg f_0(\$2) \wedge \neg f_1(\$1) \wedge \neg f_1(\$2) \wedge \neg f_2(\$0) \wedge \neg f_2(\$2) \wedge f_3(\$3)) \vee (\neg f_0(\$0) \wedge \neg f_0(\$2) \wedge \neg f_1(\$0) \wedge \neg f_1(\$1) \wedge \neg f_2(\$0) \wedge \neg f_2(\$1) \wedge \neg f_2(\$2) \wedge f_3(\$3)) \vee (\neg f_0(\$0) \wedge \neg f_0(\$1) \wedge \neg f_1(\$1) \wedge \neg f_1(\$2) \wedge \neg f_2(\$0) \wedge \neg f_2(\$1) \wedge \neg f_2(\$2) \wedge f_3(\$3)) \vee (\neg f_0(\$0) \wedge \neg f_0(\$1) \wedge \neg f_1(\$0) \wedge \neg f_1(\$1) \wedge \neg f_2(\$2) \wedge f_3(\$3)) \vee (\neg f_0(\$0) \wedge \neg f_0(\$1) \wedge \neg f_1(\$0) \wedge \neg f_1(\$1) \wedge \neg f_1(\$2) \wedge \neg f_2(\$0) \wedge \neg f_2(\$2) \wedge f_3(\$3)) \vee (\neg f_0(\$0) \wedge \neg f_0(\$2) \wedge \neg f_1(\$1) \wedge \neg f_2(\$0) \wedge \neg f_2(\$2) \wedge f_3(\$3)) \vee (\neg f_0(\$0) \wedge \neg f_0(\$2) \wedge \neg f_1(\$0) \wedge \neg f_1(\$1) \wedge \neg f_1(\$2) \wedge \neg f_2(\$1) \wedge \neg f_2(\$2) \wedge f_3(\$3)) \vee \dots,$$

and contains 24 clauses at the top level with 144 literals in total.

Example 11 (Mastermind (alternative)). For completeness, we show another way to formalize the Mastermind problem, which does not need algorithmic generation of the formulas. Let $\mathcal{M'}_{n,m} = (X, \varphi_0, T, \Sigma, E, F, \Phi)$.

- X and φ_0 is defined as in Example 10.
- $T = \{g_{k_1,\dots,k_m} \mid k_i \in \{1, \dots, n\}, \sum_i k_i = n\}$,
  $\Sigma = C$,
  $E = \{(g_{k_1,\dots,k_m}, p) \mid p \in \Sigma^n, |\{i \mid p[i] = j\}| = k_j\}$.
  The type $g_{k_1,\dots,k_m}$ covers all the guesses in which the number of j-colored pegs is k_j. Therefore, two guesses for which we use the same pegs (pegs are just shuffled) are of the same type, but if we change a peg for one with different color, it is other type of experiment.
- $F = \{f_1, \dots, f_n\}$, where $f_i(c) = x_{i,c}$ for $1 \leq i \leq n$,

$$\Phi(g_{k_1,\ldots,k_n}) = \Big\{$$
$$\texttt{Exactly}_b \{f_i(\$i) \mid 1 \le i \le n\} \wedge \tag{1}$$
$$\texttt{Exactly}_t \bigcup \big\{\{\texttt{AtLeast}_l (x_{1,j},\ldots,x_{n,j}) \mid 1 \le l \le k_j\} \mid 1 \le j \le m\big\} \tag{2}$$
$$\mid 0 \le b \le t, 0 \le t \le n\Big\}.$$

Part (1) of the formula captures the number of the black markers. Part (2) captures the total number of markers. Indeed, we get k markers for color j if and only if $k < k_j$ and there are at least k pegs of color j in the code, i.e. all the formulas $\texttt{AtLeast}_i (x_{1,j},\ldots,x_{n,j})$ are satisfied for $i \le k$. Note that since the number of pegs of each color is fixed by the type and we do not care about the exact positions, this part of the formula is not parametrized.

We do not provide the formal definition of other Code breaking Games presented in Chapter 2. However, a computer language for game specification that is based on this formalism is introduced in Chapter 4, and definition of all the games in this language can be found in ??.

3.3 Strategies

Definition 12 (Strategy). A *strategy* is a function $\sigma : \Omega^* \to E$, determining the next experiment for a given finite solving process.

A strategy σ together with a valuation $v \in \texttt{Val}^*$ induce an infinite solving process

$$\lambda_v^\sigma = e_1, \varphi_1, e_2, \varphi_2, \ldots,$$

where $e_{i+1} = \sigma(e_1, \varphi_1, \ldots, e_i, \varphi_i)$ and $\varphi_{i+1} \in \Phi(e_{i+1})$ is such that $v(\varphi_{i+1}) = 1$, for all $i \in \mathbb{N}$. Note that thanks to the well-formed property, there is always exactly one such φ_{i+1}.

We define *length* of a strategy σ on a valuation v, denoted $|\sigma|_v$, as the smallest $k \in \mathbb{N}_0$ such that $\lambda_v^\sigma\langle k\rangle$ uniquely determines the code, i.e.

$$|\sigma|_v = \min \{k \in \mathbb{N}_0 \mid \#\lambda_v^\sigma\langle k\rangle = 1\}$$

The *worst-case number of experiments* Λ^σ of a strategy σ is the maximal length of the strategy on a valuation v, over all models v of φ_0, i.e.

$$\Lambda^\sigma = \max_{v \in \texttt{Val}^*} |\sigma|_v.$$

We say that a strategy σ *solves the game* if Λ^σ is finite. The game is *soluble* if there exists a strategy that solves the game.
The *average-case number of experiments* $\Lambda^\sigma_{\text{exp}}$ of a strategy σ is the expected number of experiments if the code is selected from models of φ_0 with uniform distribution, i.e.

$$\Lambda^\sigma_{\text{exp}} = \frac{\sum_{v \in \mathtt{Val}^*} |\sigma|_v}{\#\varphi_0}.$$

Definition 13 (Optimal strategy). A strategy σ is *worst-case optimal* if $\Lambda^\sigma \leq \Lambda^{\sigma'}$ for any strategy σ'. A strategy σ is *average-case optimal* if $\Lambda^\sigma_{\text{exp}} \leq \Lambda^{\sigma'}_{\text{exp}}$ for any strategy σ'.

Lemma 14. *Let $b = \max_{t \in T} |\Phi(t)|$ be the maximal number of possible outcomes of an experiment. Then for every strategy σ,*

$$\Lambda^\sigma \geq \lceil \log_b(\#\varphi_0) \rceil.$$

Proof. Let us fix a strategy σ and $k = \Lambda^\sigma$. For an unknown model v of φ_0, $\lambda^\sigma_v\langle k \rangle$ can take up to b^k different values. By pidgeon-hole principle, if $\#\varphi_0 > b^k$, there must be a valuation v such that $\#\lambda^\sigma_v\langle k \rangle > 1$. This would be a contradiction with $k = \Lambda^\sigma$ and, therefore, $\#\varphi_0 \leq b^k$, which is equivalent with the statement of the lemma. ■

Lemma 15. *Let σ be a strategy and let $v_1, v_2 \in \mathtt{Val}^*$. If v_1 is a model of $\lambda^\sigma_{v_2}\langle k \rangle$, then $\lambda^\sigma_{v_1}[1..k] = \lambda^\sigma_{v_2}[1..k]$.*

Proof. Let $\lambda_1 = \lambda^\sigma_{v_1}$, $\lambda_2 = \lambda^\sigma_{v_2}$ and consider the first place where λ_1 and λ_2 differs. It cannot be an experiment $\lambda_1(i) \neq \lambda_2(i)$ as they are both values of the same strategy on the same process: $\lambda_1(i) = \sigma(\lambda_1[1..i-1]) = \sigma(\lambda_2[1..i-1]) = \lambda_2(i)$.
Suppose it is an outcome of the i-th experiment, $\lambda_1[i] \neq \lambda_2[i]$ and $i \leq k$. Since v_1 satisfies $\lambda_2\langle k \rangle$ and $i \leq k$, it satisfies $\lambda_2[i]$ as well. However, v_1 always satisfies $\lambda_1[i]$ and both $\lambda_1[i]$ and $\lambda_2[i]$ are from the set $\Phi(\lambda_1(i)) = \Phi(\lambda_2(i))$. Since there is exactly one satisfied experiment for each valuation in the set, $\lambda_1[i]$ and $\lambda_2[i]$ must be the same. Contradiction. ■

Example 16. TODO: Příklad jednoduché hry, strategie, odhadu pomocí lematu, optimální strategie.

Non-adaptive strategies

Definition 17 (Non-adaptive strategy). A strategy σ is *non-adaptive* if it decides the next experiment based on the length of the solving process only, i.e. whenever λ_1 and λ_2 are processes such that $|\lambda_1| = |\lambda_2|$, then $\sigma(\lambda_1) = \sigma(\lambda_2)$.
Non-adaptive strategies can be seen as functions $\tau : \mathbb{N}_0 \to E$. Then $\sigma(\lambda) = \tau(|\lambda|)$.

Non-adaptive strategies corresponds to the well studied problems of static mastermind and non-adaptive strategies for the coutnerfeit coin problem **??** **??**. We mention them here just to show the possibility of formulating these problems in our framework but we do not study them any further.

Memory-less strategies

Definition 18 (Memory-less strategy). A strategy σ is *memory-less* if it decides the next experiment based on the accumulated knowledge only, i.e. whenever λ_1 and λ_2 are processes such that if $\lambda_1\langle\rangle \equiv \lambda_2\langle\rangle$ then $\sigma(\lambda_1) = \sigma(\lambda_2)$.
Memory-less strategies can be considered as functions $\tau : \mathtt{Form}^* \to E$ such that $\varphi_1 \equiv \varphi_2 \Rightarrow \tau(\varphi_1) = \tau(\varphi_2)$. Then $\sigma(\lambda) = \tau(\lambda\langle\rangle)$.

Lemma 19. *Let σ be a memory-less strategy and $v \in \mathtt{Val}^*$. If there exists $k \in \mathbb{N}$ such that $\#\lambda_v^\sigma\langle k\rangle = \#\lambda_v^\sigma\langle k+1\rangle$, then $\#\lambda_v^\sigma\langle k\rangle = \#\lambda_v^\sigma\langle k+l\rangle$ for any $l \in \mathbb{N}$.*

Proof. For the sake of simplicity, let $\alpha^k = \lambda_v^\sigma\langle k\rangle$. There is a formula $\varphi \in \Phi(\alpha^k)$, such that $\alpha^{k+1} \equiv \alpha^k \wedge \varphi$. Therefore, if α^{k+1} is satisfied by valuation v, so must be α^k. Since $\#\alpha^k = \#\alpha^{k+1}$, the sets of valuations satisfying α^k and α^{k+1} are exactly the same and the formulas are thus equivalent. This implies $\sigma(\alpha^k) = \sigma(\alpha^{k+1})$ and $\alpha^{k+2} \equiv \alpha^{k+1} \wedge \varphi \equiv \alpha^{k+1}$.
By induction, $\sigma(\alpha^{k+l}) = \sigma(\alpha^k)$ and $\alpha^{k+l} \equiv \alpha^k$ for any $l \in \mathbb{N}$. ∎

Lemma 20. *Let σ be a strategy. Then there exists a memory-less strategy τ such that $|\sigma|_v \geq |\tau|_v$ for all $v \in \mathtt{Val}^*$.*

Proof. Let us choose any total order $\varphi_1, \varphi_2, \ldots$ of $\mathtt{Form}^*$ such that if φ_i implies φ_j, then $i \leq j$. We build a sequence of strategies $\sigma_0, \sigma_1, \sigma_2, \ldots$ inductively in the following way. Let $\sigma_0 = \sigma$.

- If there is no $v \in \mathtt{Val}^*, k \in \mathbb{N}_0$ such that $\lambda_v^{\sigma_{i-1}}\langle k\rangle \equiv \varphi_i$, select any $e \in E$ and define σ_i by
$$\sigma_i(\lambda) = \begin{cases} \sigma_{i-1}(\lambda) & \text{if } \lambda\langle\rangle \not\equiv \varphi_i, \\ e & \text{if } \lambda\langle\rangle \equiv \varphi_i. \end{cases}$$
Clearly, all induced solving processes for σ_i and σ_{i-1} are the same and $|\sigma_i|_v = |\sigma_{i-1}|_v$.

- If there exists $v \in \mathtt{Val}^*, k \in \mathbb{N}_0$ such that $\lambda_v^{\sigma_{i-1}}\langle k\rangle \equiv \varphi_i$, choose the largest l such that $\lambda_v^{\sigma_{i-1}}\langle l\rangle \equiv \varphi_i$ and define

$$\sigma_i(\lambda) = \begin{cases} \sigma_{i-1}(\lambda) & \text{if } \lambda\langle\rangle \not\equiv \varphi_i, \\ \lambda_v^{\sigma_{i-1}}(l) & \text{if } \lambda\langle\rangle \equiv \varphi_i. \end{cases}$$

First we prove that this definition is correct. Let v_1, v_2, k_1, k_2 be such that $\lambda_{v_1}^{\sigma_{i-1}}\langle k_1\rangle \equiv \varphi_i \equiv \lambda_{v_2}^{\sigma_{i-1}}\langle k_2\rangle$. Take l_1, l_2 as the largest numbers such that $\lambda_{v_1}^{\sigma_{i-1}}\langle l_1\rangle \equiv \varphi_i \equiv \lambda_{v_2}^{\sigma_{i-1}}\langle l_2\rangle$. Since v_1 satisfies $\lambda_{v_2}^{\sigma_{i-1}}\langle l_2\rangle \equiv \varphi_i$, then $\lambda_{v_2}^{\sigma_{i-1}}[1..l_2] = \lambda_{v_1}^{\sigma_{i-1}}[1..l_2]$ by Lemma 15. The same holds for l_1 which means that $l_1 = l_2$ and $\lambda_{v_1}^{\sigma_{i-1}}(l_1) = \lambda_{v_1}^{\sigma_{i-1}}(l_2)$, which proves that the definition of σ_i is independent of the exact choices of v and k.
Now $|\sigma_i|_v = |\sigma_{i-1}|_v - (l-k)$, where k and l is the smallest and the largest number such that $\lambda_v^{\sigma_{i-1}}\langle k\rangle \equiv \varphi_i$ and $\lambda_v^{\sigma_{i-1}}\langle l\rangle \equiv \varphi_i$, respectively, because $\lambda_v^{\sigma_{i-1}}(l) = \lambda_v^{\sigma_i}(k)$ and due to the ordering, the rest of the process is independent of the beginning.

The last strategy of the sequence is clearly memory-less and satisfies the condition in the lemma. ∎

Definition 21 (Greedy strategy). Let $f : \mathtt{Form}_X \to \mathbb{Z}$. A memory-less strategy σ is *f-greedy* if for every $\varphi \in \mathtt{Form}_X$ and $e' \in E$,

$$\max_{\substack{\psi\in\Phi(\sigma(\varphi)) \\ SAT(\varphi\wedge\psi)}} f(\varphi\wedge\psi) \le \max_{\substack{\psi\in\Phi(e) \\ SAT(\varphi\wedge\psi)}} f(\varphi\wedge\psi).$$

In words, a greedy strategy minimizes the value of f on the formula in the next step. We say σ is *greedy* if it is $\#_X$-greedy.

Lemma 22. *Let $b = \max_{t\in T} |\Phi(t)|$ be the maximal number of possible outcomes of an experiment. If for any $\varphi \in \mathtt{Form}^*$,*

$$\exists e. \max_{\psi\in\Phi(e)} \#(\varphi\wedge\psi) = \left\lceil \frac{\#\varphi}{b} \right\rceil,$$

then a greedy strategy σ is optimal and

$$\Lambda^\sigma = \lceil \log_b(\#\varphi_0) \rceil.$$

Proof. TODO: Napsat důkaz.

Example 23. Greedy strategies are optimal in the fake-coin game $\mathcal{F}_n$.
TODO: Napsat důkaz.

3.4 Symmetries in Code Braking Games

Definition 24 (Symmetric experiment). For an experiment $e = (t, p)$ and a permutation $\pi \in \texttt{Perm}_X$, a π-symmetric experiment $e^\pi = (t, p') \in E$ is an experiment of the same type such that $\{\varphi^\pi \in \Phi(e)\} = \{\varphi \in \Phi(e^\pi)\}$. Clearly, no such experiment may exists.

Definition 25 (Symmetry group). We define a *symmetry group* Π as the maximal subset of $\texttt{Perm}_X$ such that for every $\pi \in \Pi$ and for every experiment $e \in E$, there exists a π-symmetric experiment e^π.

Definition 26 (Consistent strategy). A memory-less strategy σ is *consistent* if and only if for every $\varphi \in \texttt{Form}_X$ and every $\pi \in \Pi$, there exists $\rho \in \Pi$ such that $\varphi^\pi \equiv \varphi^\rho$ and $\sigma(\varphi^\rho) = \sigma(\varphi)^\rho$.

Lemma 27. *Let σ be a memory-less strategy. There exists a consistent memory-less strategy τ such that $|\sigma|_v \geq |\tau|_v$ for all $v \in \texttt{Val}_X$ satisfying φ_0.*

Proof.

Definition 28 (Experiment equivalence). An experiment $e_1 \in E$ is equivalent to $e_2 \in E$ with respect to φ, written $e_1 \cong_\varphi e_2$, if and only if there exists a permutation $\pi \in \Pi$ such that $\{\varphi \wedge \psi \mid \psi \in \Phi(e_1)\} \equiv \{(\varphi \wedge \psi)^\pi \mid \psi \in \Phi(e_2)\}$.

Theorem 29. *Let σ, τ be two consistent memory-less strategies, such that $\sigma(\varphi) \cong_\varphi \tau(\varphi)$ for any $\varphi \in \texttt{Form}_X$. There is a bijection $f : \texttt{Val}_X \to \texttt{Val}_X$ such that $|\sigma|_v = |\tau|_{f(v)}$.*

Proof.

Corollary 30. *Let σ_1, σ_2 be two consistent memory-less strategies, such that $\sigma_1(\varphi) \cong_\varphi \sigma_2(\varphi)$ for any $\varphi \in \texttt{Form}_X$. Then $\Lambda^{\sigma_1} = \Lambda^{\sigma_2}$ and $\Lambda^{\sigma_1}_{exp} = \Lambda^{\sigma_2}_{exp}$.*

3.5 Symmetry Breaking

Phase 1 - Interchangeble symbols

Phase 2 - Canonical Form of parametrization

Phase 3 - Canonical Form of formula graph

Comparison

4 COBRA - COde BReaking Game Analyzer

4.1 Input language

```
       <code> ::= <line> | <code> <line>
       <line> ::= VARIABLE ident | VARIABLES <ident-list> |
                  RESTRICTION <formula> | ALPHABET <string-list> |
                  MAPPING ident <ident-list> | EXPERIMENT string int |
                  PARAMS-DISTINCT <int-list> | PARAMS-SORTED <int-list> |
                  OUTCOME string <formula>
    <formula> ::= ident | ( <formula> ) | ! <formula> |
                  <formula> AND <formula> | <formula> OR <formula> |
                  AND( <formula-list> ) | OR( <formula-list> ) |
                  <formula> → <formula> | <formula> ← <formula> |
                  <formula> ↔ <formula> | ident ($ int ) |
                  ATLEAST- int ( <formula-list> ) |
                  ATMOST- int ( <formula-list> ) |
                  EXACTLY- int ( <formula-list> )
 <ident-list> ::= ident | <ident-list> , ident
   <int-list> ::= int | <int-list> , int
<formula-list> ::= <formula> | <formula-list> , <formula>
<string-list> ::= string | <string-list> , string
```

4.2 Modes

4.3 Implementation details

Programming Language and Style

Since the problem we are trying to solve is very computationally demanding, we had to choose a high-performing programming language. The tools we want to use, especially SAT solvers, are typically written in C/C++, so C++ was a natural choice for our tool. Cobra is written in the latest standard of ISO C++, namely C++11, which contains significant changes both in the language and in the standard libraries and, in our opinion, improves readability compared to previous versions.

We wanted the style of our code to be consistent and to usage of the language in the best manner possible according to industrial practice. From the wide range of style guides available online we chose *Google C++ Style Guide*?? and made the code compliant with all its rules except for a few exception. The only significant one of those are lambda functions, which are forbidden by the style guide due to various reasons??, but we think they are more beneficial than harmful in this project.

Compiler Requirements

The usage of a modern standard requires a modern compiler, which supports all the C++11 feautres we use. We recommend using starndard `gcc`; you need version 4.8 or higher. For `clang`, you need version 3.2 or higher.
The tool is platform independent. We tested compilation and functionality on all three major operating systems, on Linux (Ubuntu 12.04), Mac OS X (10.9) and Windows (8.1).

Unit testing.

Unit testing has became a common part of software development process in the recent years. Correctness was a top priority during the development and unit tests are a perfect way to capture potential programmer's error as soon as possible and avoid regression.
There is a lot of unit tests framework for C++. We focused on simplicity, minimal amount of work needed to add new tests and good assertion support, and opted for *Google Test*??.
All available tests are compiled and excecuted if you run `make test` in the root folder. This should serve as a basic sanity test and we highly recommend doing this in case anyone needs to change something in the code.

4.4 Sat solving

5 Conclusions

Bibliography

[1] Howard D Grossman. The twelve-coin problem. *Scripta Mathematica*, 11:360–363, 1945.

[2] Freeman J Dyson. 1931. the problem of the pennies. *The Mathematical Gazette*, pages 231–234, 1946.

[3] Richard K Guy and Richard J Nowakowski. Coin-weighing problems. *American Mathematical Monthly*, pages 164–167, 1995.

[4] Ratko Tošić. Two counterfeit coins. *Discrete Mathematics*, 46(3):295–298, 1983.

[5] Anping Li. Three counterfeit coins problem. *Journal of Combinatorial Theory, Series A*, 66(1):93–101, 1994.

[6] László Pyber. How to find many counterfeit coins? *Graphs and Combinatorics*, 2(1):173–177, 1986.

[7] Xiao-Dong Hu, PD Chen, and Frank K. Hwang. A new competitive algorithm for the counterfeit coin problem. *Information Processing Letters*, 51(4):213–218, 1994.

[8] Martin Aigner and Anping Li. Searching for counterfeit coins. *Graphs and Combinatorics*, 13(1):9–20, 1997.

[9] Axel Born, Cor AJ Hurkens, and Gerhard J Woeginger. How to detect a counterfeit coin: Adaptive versus non-adaptive solutions. *Information processing letters*, 86(3):137–141, 2003.

[10] Andrzej Pelc. Searching games with errors—fifty years of coping with liars. *Theoretical Computer Science*, 270(1):71–109, 2002.

[11] A Pelc. Detecting a counterfeit coin with unreliable weighings. *Ars Combinatoria*, 27:181–192, 1989.

[12] Annalisa De Bonis, Luisa Gargano, and Ugo Vaccaro. Optimal detection of a counterfeit coin with multi-arms balances. *Discrete applied mathematics*, 61(2):121–131, 1995.

[13] Tanya Khovanova. Parallel weighings. *arXiv preprint arXiv:1310.7268*, 2013.