

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 Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.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, mastermind, counterfeit coin, sat solving, greedy strategy



# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Code Breaking Games</b>	<b>5</b>
2.1	<i>The Counterfeit Coin</i>	5
2.2	<i>Mastermind</i>	5
2.3	<i>Other Problems and Applications</i>	6
<b>3</b>	<b>General model</b>	<b>7</b>
3.1	<i>Notation and Terminology</i>	7
3.2	<i>Formal definition</i>	7
3.3	<i>Strategies</i>	12
3.4	<i>Symmetries in Code Braking Games</i>	15
3.5	<i>Symmetry Breaking</i>	17
<b>4</b>	<b>COBRA - COntributed BReaking Game Analyzer</b>	<b>19</b>
4.1	<i>Input language</i>	19
4.2	<i>Modes</i>	19
4.3	<i>Implementation details</i>	19
4.4	<i>Sat solving</i>	20
<b>5</b>	<b>Conclusions</b>	<b>21</b>





# 1 Introduction



## 2 Code Breaking Games

### 2.1 The Counterfeit Coin

There is a lot of variants of a logic puzzle with coins and a pair of scales balance. Here we present the most interesting ones and their generalization, which we study in the sequel.

In all the problems, you can use the scales only to weight coins. You can put as many coins at the sides as you like as long as the number is the same. All information you get is that both sides weight equally or which side is heavier (i.e., there are 3 possible results).

The weight of a *fake* coin is always different than the weight of a authentic one but it is not know whether it is heavier or lighter.

**Problem 1 (The twelve coin problem).** *You are given 12 coins, exactly one of which is fake. Determine the unique coin and its weight relative to others. You can use the balance at most 3 times.*

**Problem 2 (The thirteen coin problem).** *You are given 13 coins, exactly one of which is fake. You have one more coin at your disposal which is guaranteed to be authentic. Determine the unique fake coin and its weight relative to others.*

**Problem 3 (General fake coin problem).** *You are given  $n$  coins,  $f$  of them are fake (some of them may be lighter, some of them heavier). You have another  $m$  authentic coins at your disposal. In as less weightings as possible, determine which coins are fake.*

### 2.2 Mastermind

*Mastermind* is a classic 2-player board game invented by Mordecai Meirowitz in 1970[wiki]. The principle of the game is the same as of *Bulls and Cows*, it just uses colors instead of letters.

## 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  $\text{Form}_X$  be the set of all propositional formulas over the set of variables  $X$ ;  $\text{Val}_X$  be the set of all valuations (boolean interpretation) of variables  $X$ . Formulas  $\varphi_0, \varphi_1 \in \text{Form}_X$  are (semantically) equivalent, written  $\varphi_0 \equiv \varphi_1$ , if  $v(\varphi_0) = v(\varphi_1)$  for all  $v \in \text{Val}_X$ . We say that  $v$  is a *model* of  $\varphi$  or that  $v$  *satisfies*  $\varphi$  if  $v(\varphi) = 1$ .

For a formula  $\varphi \in \text{Form}_X$ , let  $\#_X \varphi = |\{v \in \text{Val}_X \mid v(\varphi) = 1\}|$  be the number of models of  $\varphi$  (valuations satisfying  $\varphi$ ). We often omit the index  $X$  if it is clear from the context.

The set of all permutations of a set  $X$  (bijections  $X \rightarrow X$ ) is denoted by  $\text{Perm}_X$  and  $\text{id}_X$  is the identity permutation.

### 3.2 Formal definition

Code breaking games are games between two players – a *codemaker* and a *codebreaker*. First, the codemaker chooses a secret code. Then the codebreaker successively performs experiments and collects partial information about the code, which is given by the codemaker according to the rules of the game. The codebreaker strives to reveal the code in minimal number of experiments. ... (introduction)

Within the framework of propositional logic, we represent the secret code as a valuation of propositional variables. The game can be 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,
- $\varphi_0 \in \text{Form}_X$  is a satisfiable propositional formula,
- $T$  is a finite set of types of experiments,

- $\Sigma$  is a finite alphabet,
- $E \subseteq T \times \Sigma^*$  is an *experiment* relation, and
- $F$  is a finite collection of functions of type  $\Sigma \rightarrow X$ ,
- $\Phi : T \rightarrow 2^{\text{PForm}_{X,F,\Sigma}}$  is an *outcome function* such that  $\Phi(t)$  is finite for any  $t \in T$ . Definition of  $\text{PForm}$  follows (Definition 5).

**Definition 5 (Parametrized formula).** A set of *parametrized formulas*  $\text{PForm}_{X,F,\Sigma}$  is a set of all strings  $\psi$  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  $p \in \Sigma^*$  on a formula  $\psi$ , which is defined recursively on the structure of  $\psi$  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  $f(\$n)$  apperas in  $\psi$  and  $|p| < n$ , then  $\psi(p)$  is undefined. **TODO: Or false? Does it matter?**

...

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  $(e, \varphi)$  such that  $\varphi \in \Phi(e)$ . Let us denote the set of evaluated experiments by  $\Omega$ .

A *solving process* is a finite or infinite sequence of evaluated experiments.

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

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

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,
- $\lambda[1..k] = e_1, \varphi_1, \dots, e_k, \varphi_k$  denote the prefix of length  $k$ , and
- $\lambda\langle k \rangle = \varphi_0 \wedge \varphi_1 \wedge \dots \wedge \varphi_k$  denote the accrued knowledge after the first  $k$  experiments (including the initial restriction  $\varphi_0$ ). For finite  $\lambda$ , let  $\lambda\langle \rangle = \lambda(|\lambda|)$  be the overall accrued knowledge.

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

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

As the semantics of non-well-formed games is unclear, we focus only on well-formed games and, by 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  $\mathcal{F}_n = (X, \varphi_0, T, \Sigma, E, F, \Phi)$ .

- $X = \{x_1, x_2, \dots, x_n, y\}$ ,  
 $\varphi_0 = \mathbf{Exactly}_1 \{x_1, \dots, x_n\}$ .

Intuitively, variable  $x_i$  tells weather the coin  $i$  is fake. Variable  $y$  tells weather it is lighter or heavier. Formula  $\varphi_0$  says that exactly one coin is fake.

- $T = \{w_2, w_4, \dots, w_n\}$ ,  
 $\Sigma = \{1, 2, \dots, n\}$ ,

$$E = \bigcup_{1 \leq m \leq n/2} \{(w_{2m}, p) \mid p \in \{1, \dots, n\}^{2m}, \forall x \in X. \#_x(p) \leq 1\}.$$

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  $1 \leq i \leq n$ ,  
 $\Phi(w_m) =$   
 $\{((f_x(\$1) \vee \dots \vee f_x(\$m)) \wedge \neg y) \vee ((f_x(\$m+1) \vee \dots \vee f_x(\$2m)) \wedge y),$   
 $((f_x(\$1) \vee \dots \vee f_x(\$m)) \wedge y) \vee ((f_x(\$m+1) \vee \dots \vee f_x(\$2m)) \wedge \neg y),$   
 $\neg(f_x(\$1) \vee \dots \vee f_x(\$2m))\}.$

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.  $\mathcal{F}'_n = (X, \varphi_0, T, \Sigma, E, F, \Phi)$ .

- $X = \{x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n\}$ ,  
 $\varphi_0 = \text{Exactly}_1 \{x_1, \dots, x_n, y_1, \dots, y_n\}.$   
Variable  $x_i$  tells that the coin  $i$  is lighter, variable  $y_i$  tells that the coin  $i$  is heavier. Formule  $\varphi_0$  says that exactly one coin is different.
- $T, \Sigma, E$  is defined as in [Example 8](#).
- $F = \{f_x, f_y\}$ , where  $f_x(i) = x_i$  and  $f_y(i) = y_i$  for  $1 \leq i \leq n$ ,  
 $\Phi(w_m) = \{(f_x(\$1) \vee \dots \vee f_x(\$m)) \vee (f_y(\$m+1) \vee \dots \vee f_y(\$2m)),$   
 $(f_y(\$1) \vee \dots \vee f_y(\$m)) \vee (f_x(\$m+1) \vee \dots \vee f_x(\$2m)),$   
 $\neg(f_x(\$1) \vee \dots \vee f_x(\$2m) \vee f_y(\$1) \vee \dots \vee f_y(\$2m))\}.$

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

- $X = \{x_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m\},$   
 $\varphi_0 = \bigwedge \{\text{Exactly}_1 \{x_{i,j} \mid 1 \leq j \leq m\} \mid 1 \leq i \leq n\}.$   
Variable  $x_{i,j}$  tells whether there is the color  $j$  at position  $i$ . Formula  $\varphi_0$  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 is any sequence of colors of length  $n$ .



- $F = \{f_1, \dots, f_n\}$ , 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  $P = \{1, 2, \dots, n\}$  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 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 **TODO: ??** clauses at the top level.

**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  $\varphi_0$  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  $1 \leq i \leq n$ ,

$$\Phi(g_{k_1, \dots, k_n}) = \left\{ \begin{array}{l} \text{Exactly}_b \{f_i(\$i) \mid 1 \leq i \leq n\} \wedge \\ \text{Exactly}_t \bigcup \{ \{ \text{AtLeast}_l(x_{1,j}, \dots, x_{n,j}) \mid 1 \leq l \leq k_j \} \mid 1 \leq j \leq m \} \\ \mid 0 \leq b \leq t, 0 \leq t \leq n \}. \end{array} \right. \quad (1)$$

$$\text{Exactly}_t \bigcup \{ \{ \text{AtLeast}_l(x_{1,j}, \dots, x_{n,j}) \mid 1 \leq l \leq k_j \} \mid 1 \leq j \leq m \} \quad (2)$$

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  $\text{AtLeast}_i(x_{1,j}, \dots, x_{n,j})$  are satisfied for  $i \leq k$ . 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  $\sigma : \Omega^* \rightarrow E$ , determining the next experiment for a given finite solving process.

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

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

where  $e_{i+1} = \sigma(e_1, \varphi_1, \dots, e_i, \varphi_i)$  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  $\varphi_{i+1}$ .

We define *length* of a strategy  $\sigma$  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*  $\Lambda^\sigma$  of a strategy  $\sigma$  is the maximal length of the strategy on a valuation  $v$ , over all models  $v$  of  $\varphi_0$ , i.e.

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

We say that a strategy  $\sigma$  *solves the game* if  $\Lambda^\sigma$  is finite. The game is *solvable* if there exists a strategy that solves the game.

The *average-case number of experiments*  $\Lambda_{\text{exp}}^\sigma$  of a strategy  $\sigma$  is the expected number of experiments if the code is selected from models of  $\varphi_0$  with uniform distribution, i.e.

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

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

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

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

*Proof.* Let us fix a strategy  $\sigma$  and  $k = \Lambda^\sigma$ . For an unknown model  $v$  of  $\varphi_0$ ,  $\lambda_v^\sigma(k)$  can take up to  $b^k$  different values. By pigeon-hole principle, if  $\#\varphi_0 > b^k$ , there must be a valuation  $v$  such that  $\#\lambda_v^\sigma(k) > 1$ . 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  $\sigma$  be a strategy and let  $v_1, v_2 \in \text{Val}^*$ . If  $v_1$  is a model of  $\lambda_{v_2}^\sigma(k)$ , then  $\lambda_{v_1}^\sigma[1..k] = \lambda_{v_2}^\sigma[1..k]$ .

*Proof.* Let  $\lambda_1 = \lambda_{v_1}^\sigma$ ,  $\lambda_2 = \lambda_{v_2}^\sigma$  and consider the first place where  $\lambda_1$  and  $\lambda_2$  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  $\lambda_2(k)$  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: ...

### Non-adaptive strategies

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

Non-adaptive strategies corresponds to the well studied problems of static mastermind and non-adaptive strategies for the counterfeit 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  $\sigma$  is *memory-less* if it decides the next experiment based on the accumulated knowledge only, i.e. whenever  $\lambda_1$  and  $\lambda_2$  are processes such that  $\lambda_1 \langle |\lambda_1| \rangle \equiv \lambda_2 \langle |\lambda_2| \rangle$  then  $\sigma(\lambda_1) = \sigma(\lambda_2)$ . Memory-less strategies can be considered as functions  $\tau : \mathbf{Form}^* \rightarrow E$  such that  $\varphi_1 \equiv \varphi_2 \Rightarrow \tau(\varphi_1) = \tau(\varphi_2)$ . Then  $\sigma(\lambda) = \tau(\lambda \langle \rangle)$ .

**Lemma 19.** Let  $\sigma$  be a memory-less strategy and  $v$  a model of  $\varphi_0$ . 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  $\alpha^{k+1}$  is satisfied by valuation  $v$ , so must be  $\alpha^k$ . Since  $\# \alpha^k = \# \alpha^{k+1}$ , the sets of valuations satisfying  $\alpha^k$  and  $\alpha^{k+1}$  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  $\sigma$  be a strategy. Then there exists a memory-less strategy  $\tau$  such that  $|\sigma|_v \geq |\tau|_v$  for all  $v$  satisfying  $\varphi_0$ .

*Proof.* Let us choose any total order  $\varphi_1, \varphi_2, \dots$  of  $\mathbf{Form}^*$  such that if  $\varphi_i$  implies  $\varphi_j$ , then  $i \leq j$ . We build a sequence of strategies  $\sigma_0, \sigma_1, \sigma_2, \dots$  in the following way: Let  $\sigma_0 = \sigma$  and for  $i > 0$ ,

$$\sigma_i(\lambda) = \begin{cases} \sigma_{i-1}(\lambda) & \text{if } \lambda \langle \rangle \not\equiv \varphi_i \\ \lambda_{v_i}^{\sigma_{i-1}}(k_i + 1) & \text{if } \lambda \langle \rangle \equiv \varphi_i, \text{ where } (v_i, k_i) = \arg \min_K |\sigma_{i-1}|_v - k, \\ & \text{and } K = \{(v, k) \in \mathbf{Val} \times \mathbb{N}_0 \mid \lambda_v^{\sigma_{i-1}} \langle k \rangle \equiv \varphi_i\}. \end{cases}$$

We prove that  $|\sigma_i|_v \leq |\sigma_{i-1}|_v$ . If there is no  $k$  such that  $\lambda_v^{\sigma_i} \langle k \rangle \equiv \varphi_i$  then the processes  $\lambda_v^{\sigma_i}$  and  $\lambda_v^{\sigma_{i-1}}$  are the same. If there is such  $k$ , then, due to the order of the formulas, the following choices of  $\sigma$  depend only on the accrued knowledge and, therefore,  $|\sigma_i|_v = k + |\sigma_{i-1}|_{v_i} - k_i \leq |\sigma_{i-1}|_v$  because of the choice of  $v_i, k_i$ .

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

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

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

In words, a greedy strategy minimizes the value of  $f$  on the formula in the next step. We say  $\sigma$  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  $\varphi \in \text{Form}^*$ ,

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

then a greedy strategy  $\sigma$  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  $\pi \in \text{Perm}_X$ , a  $\pi$ -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 exist.

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

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

TODO: Example, na kterém bude vidět, že jednoduchá definice nevyhovuje, protože můžu vzít symetrie  $\varphi$  a dostanu, že to má dávat různé věci.

**Lemma 27.** Let  $\sigma$  be a memory-less strategy. There exists a consistent memory-less strategy  $\tau$  such that  $|\sigma|_v \geq |\tau|_v$  for all  $v \in \text{Val}_X$  satisfying  $\varphi_0$ .

*Proof.* Similar to the proof of Lemma ???. Let us choose any total order  $\varphi_1, \varphi_2, \dots$  of  $\text{Form}_X$  such that if  $\varphi_i$  implies  $\varphi_j$ , then  $i \leq j$ . We build a sequence of strategies  $\sigma_0, \sigma_1, \sigma_2, \dots$  in the following way: Let  $\sigma_0 = \sigma$  and for  $i > 0$ ,

$$\sigma_i(\varphi) = \begin{cases} \sigma_{i-1}(\varphi) & \text{if } \nexists \pi \in \Pi. \varphi^\pi \equiv \varphi_i \\ \lambda_{v_i}^{\sigma_{i-1}}(k_i + 1) & \text{if } \exists \pi \in \Pi. \varphi^\pi \equiv \varphi_i, \text{ where } (v_i, k_i) = \arg \min_K |\sigma_{i-1}|_v - k, \\ & \text{and } K = \{(v, k) \in \text{Val} \times \mathbb{N}_0 \mid \exists \pi. \lambda_v^{\sigma_{i-1}} \langle k \rangle^\pi \equiv \varphi_i\}. \end{cases}$$

We prove that  $|\sigma_i|_v \leq |\sigma_{i-1}|_v$ . If there is no  $k$  and  $\pi$  such that  $\lambda_v^{\sigma_i} \langle k \rangle^\pi \equiv \varphi_i$  then the processes  $\lambda_v^{\sigma_i}$  and  $\lambda_v^{\sigma_{i-1}}$  are the same. If there is such  $k$  and TODO: ....

The last strategy of the sequence is consistent and satisfies the condition in the lemma. ■

**Definition 28 (Experiment equivalence).** An experiment  $e_1 \in E$  is equivalent to  $e_2 \in E$  with respect to  $\varphi$ , 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  $\sigma, \tau$  be two consistent memory-less strategies, such that  $\sigma(\varphi) \cong_\varphi \tau(\varphi)$  for any  $\varphi \in \text{Form}_X$ . There is a bijection  $f : \text{Val}_X \rightarrow \text{Val}_X$  such that  $|\sigma|_v = |\tau|_{f(v)}$ .

*Proof.* First, we prove by induction for any  $k \in \mathbb{N}_0$ , there is a permutation  $\pi \in \Pi$  such that  $(\lambda_v^\sigma \langle i \rangle)^\pi = \lambda_{v^\pi}^\tau \langle i \rangle$  for all  $i \in \mathbb{N}_0$ ,  $i \leq k$ . For better readability, let  $\alpha_k = \lambda_v^\sigma \langle k \rangle$  and  $\beta_{k,\pi} = \lambda_{v^\pi}^\tau \langle k \rangle$

For  $k = 0$ , take  $\pi = \text{id}_X$ . Clearly,  $\lambda_v^\sigma \langle 0 \rangle = \varphi_0 = \lambda_{v^{\text{id}}}^\tau \langle 0 \rangle$ .

For the induction step, suppose we have  $\pi \in \Pi$  such that  $\alpha_i^\pi = \beta_{i,\pi}$  for  $i \leq k$ . Further, suppose  $\pi$  is such that  $\sigma(\alpha_k^\pi) = \sigma(\alpha_k)^\pi$ . **TODO: Víc zdůvodnit.**

Let  $e_1 = \sigma(\alpha_k)$ ,  $e_2 = \tau(\beta_{k,\pi})$  be the  $(k+1)$ -th experiments of the strategies. It holds

$$e_2 = \tau(\beta_{k,\pi}) \cong_{\beta_{k,\pi}} \sigma(\beta_{k,\pi}) \stackrel{IH}{=} \sigma(\alpha_k^\pi) = \sigma(\alpha_k)^\pi = e_1^\pi$$

and, therefore, there exists  $\rho \in \Pi$  such that

$$\begin{aligned} \{\beta_{k,\pi} \wedge \psi \mid \psi \in \Phi(e_2)\} &= \{(\beta_{k,\pi} \wedge \psi)^\rho \mid \psi \in \Phi(e_1^\pi)\} = \\ &= \{(\alpha_k^\pi \wedge \psi^\pi)^\rho \mid \psi \in \Phi(e_1)\} = \{(\alpha_k \wedge \psi)^{\rho\pi} \mid \psi \in \Phi(e_1)\} \end{aligned} \quad (*)$$

As  $\rho \in \Pi$  and  $\Pi$  is a permutation group,  $\rho\pi \in \Pi$ .

Since the game is well-formed,  $v$  satisfies exactly one formula in  $\{\alpha_k \wedge \psi \mid \psi \in \Phi(e_1)\}$ . Therefore  $v^{\rho\pi}$  satisfies exactly one formula in  $\{(\alpha_k \wedge \psi)^{\rho\pi} \mid \psi \in \Phi(e_1)\} = \{\beta_{k,\pi} \wedge \psi \mid \psi \in \Phi(e_2)\}$ , which means that  $v^{\rho\pi}$  satisfies  $\beta_{k,\pi}$ . **TODO: Tohle nefugujeee!** From Lemma 15,  $\beta_{k,\pi} = \beta_{k,\rho\pi}$ . Both  $\alpha_{k+1}^{\rho\pi}$  and  $\beta_{k+1,\rho\pi}$  is thus the only formula from (\*) satisfied by  $v^{\rho\pi}$  and, therefore,  $\alpha_{k+1}^{\rho\pi} = \beta_{k+1,\rho\pi}$ .

Now for a fixed  $v$ , take  $k = |\sigma|_v$ , take  $\pi \in \Pi$  such that  $(\lambda_v^\sigma \langle k \rangle)^\pi = \lambda_{v^\pi}^\tau \langle k \rangle$  and define  $f(v) = v^\pi$ . Since  $(\lambda_v^\sigma \langle i \rangle)^\pi = \lambda_{f(v)}^\tau \langle i \rangle$  for  $i \leq k$  and variable permutation preserves the number of models of a formula, i.e.  $\#\varphi = \#\varphi^\pi$  for any  $\varphi \in \text{Form}_X$ ,  $\pi \in \text{Perm}_X$ , we have  $|\sigma|_v = |\tau|_{f(v)}$ .  $\blacksquare$

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

## 3.5 Symmetry Breaking

**Phase 1 - Interchangeable 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 features we use. We recommend using standard `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 become 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 executed 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