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 Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.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

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 ligther.

Problem 1 (The twelve coin problem). *You are given 12 coins, exactly one of which is fake. Determine the unique coin and its weight relavite 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 relavite 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

Let Form_X^* be the set of all prepositional formulas over the set of variables X; V_X be the set of all valuations (boolean interpretation) of variables X. Formulas $\varphi_0, \varphi_1 \in \mathrm{Form}_X^*$ are (semantically) equivalent, written $\varphi_0 \equiv \varphi_1$, if $v(\varphi_0) = v(\varphi_1)$ for all $v \in V_X$. In the whole text, we want to identify equivalent formulas. Therefore, let $\mathrm{Form}_X = \mathrm{Form}_X^* / \equiv$ and in the sequel, let us identify a formula with its corresponding equivalence class.

For a formula $\varphi \in \mathrm{Form}_X$, let $\#_X\varphi = |\{v \in V_X \mid v(\varphi) = 1\}|$ be the number of models of φ (valuations satisfying φ). This definition is correct as equivalent formulas has same number of models. We often omit the index X if it is clear from the context.

Let Perm_X be the set of all permutations of X.

3.2 Formal definition

Code breaking games are game between two players – a *codemaker* and a *codebreaker*. The codemaker selects a secret code and then evaluates the experiments performed by the codebreaker. The codebreaker chooses and performs experiments and collects partial information about the code according to some rules. 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 Counterfeitcoin 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. 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,

- $\varphi_0 \in \mathrm{Form}_X$ is a satisfiable prepositional 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 $\Sigma \to X$,
- $\Phi : T \to 2^{\mathrm{PForm}_{X,F,\Sigma}}$ is an *outcome function* such that $\Phi(t)$ is finite for any $t \in T$. Definition of PForm follows (Definition 5).

Definition 5. A set of *parametrized formulas* $\mathrm{PForm}_{X,F,\Sigma}$ 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 $\circ \in \{\wedge, \vee, =>\}$. 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. For the sake of simplicity, let us denote the set of possible outcomes for an experiment $e = (t, p) \in E$ by $\Phi(e) = \{\psi(p) \mid \psi \in \Phi(t)\}$.

Note that this 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.

Let us now describe the course of the game in the defined terms. First, the codemaker choose a valuation v of X which satisfies φ_0. Then, the codebreaker successively chooses a type $t \in T$ and a parametrization $p \in \Sigma^*$ such that $(t, p) \in E$. The codemaker gives the codebreaker a formula $\varphi \in \Phi((t, p))$ which is satisfied by the valuation v.

Here we hit a problem – so that the codemaker can always fulfill the last step, there must be a formula $\varphi \in \Phi(e)$ satisfied by any valuation. Since it might make sense to allow multiple satisfied formulas, we restrict ourselves to game where the outcome is uniquely defined for given valuation.

Definition 6. A code-breaking game is *well-formed* if for all $e \in E$,

$$\forall v \in V_X : v(\varphi_0) = 1 \Rightarrow \exists \text{ exactly one } \varphi \in \Phi(e) \,.\, v(\varphi) = 1$$

In sequel, we suppose a game to be well-formed, if not stated otherwise. Note that this property is not easy to check.

Example 7 (Fake-coin problem). Fake-coin problem with n coins, one of which is fake, can be formalized as a code breaking game $\mathcal{F}_n = (X, \varphi_0, T, \Sigma, E, F, \Phi)$.

- $X = \{x_1, x_2, \ldots, x_n, y\}$,
  $\varphi_0 =$ 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 $1 \le i \le n$,
  $\Phi(w_m) =$
  $$\big\{((f_x(\$1) \vee \ldots \vee f_x(\$m)) \wedge \neg y) \vee ((f_x(\$m+1) \vee \ldots \vee f_x(\$2m)) \wedge y),$$
  $$((f_x(\$1) \vee \ldots \vee f_x(\$m)) \wedge y) \vee ((f_x(\$m+1) \vee \ldots \vee f_x(\$2m)) \wedge \neg y),$$
  $$\neg(f_x(\$1) \vee \ldots \vee f_x(\$2m))\big\}.$$

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

- $X = \{x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n\}$,
  $\varphi_0 = \text{Exactly-1}\,\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$.
  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 7.
- $F = \{f_x, f_y\}$, where $f_x(i) = x_i$ and $f_y(i) = y_i$ for $1 \leq i \leq n$,

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

Example 9 (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 φ_0 says that there is exactly one color at each position.
- $T = \{g_{k_1,\ldots,k_m} \mid k_i \in \{1, \ldots, n\}, \sum_i k_i = n\}$,
  $\Sigma = C$,
  $E = \{(g_{k_1,\ldots,k_m}, p) \mid p \in \Sigma^n, \#i.(p[i] = j) = k_j\}$.
  The type $g_{k_1,\ldots,k_m}$ 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.
- $F = \{f_1, \ldots, f_n\}$, where $f_i(c) = x_{i,c}$ for $1 \leq i \leq n$,

$$\begin{aligned}\Phi(g_{k_1,\ldots,k_n}) = \Big\{&\\ &\text{Exactly-b}\,\{f_i(\$i) \mid 1 \leq i \leq n\} \wedge\\ &\text{Exactly-t}\,\bigcup\big\{\{\text{AtLeast-l}\,(x_{1,j}, \ldots, x_{n,j}) \mid 1 \leq l \leq k_j\} \mid 1 \leq j \leq m\big\}\\ &\qquad\qquad \mid 0 \leq b \leq t, 0 \leq t \leq n\Big\}.\end{aligned}$$

TODO: Zdůvodnit proč to funguje.

Example 10 (Mastermind (alternative)). $\mathcal{M'}_{n,m} = (X, \varphi_0, T, \Sigma, E, F, \Phi)$.

- X and φ_0 is defined as in Example 9.

- $T = \{g\}$,
  $\Sigma = C$,
  $E = \{(g, p) \mid p \in \Sigma^n\}$.

- $F = \{f_1, \ldots, f_n\}$, 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.

Consider a fixed color combination (code) and a guess. We assign a symbol $\bullet$, $\times$ or a number to each position in the following way. If the color at a position is same in the code and the guess, we assign $\bullet$ to this position and the player gets a black peg for it. If the color in the guess at position i differs but it appears appears at position j in the code, we assign j to position i and the player gets a white peg. Also, position j must not be assigned $\bullet$ and the number j must not be assigned to any other position. We assign $\times$ to all other positions. For example, if the code is 1234 and the guess is 5251, the assignment is $\underline{\times \bullet \times 1}$.
We start the computation of Outcome(b, w) with generation of all combinations of $\bullet$, $\times$ and different numbers from 1 to n, so that there is b-times $\bullet$, w-times a number and no number refers to a position with $\bullet$.
Next, for each combination, we generate a conjunction in the following way:

- For $\bullet$ at position i, we add $f_i(\$i)$.
- For a number j at position i, we add $\neg f_i(\$i) \wedge f_j(\$i)$.
- For $\times$ at position i, we add $\neg f_j(\$i)$ for any other position j with $\times$.

The result is a disjunction of all these clauses, which effectively enumerates all the cases.
To get a better idea about the results, this is Outcome$(1, 1)$ for n = 4:

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

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

However complicated this may look, note that it is not much different from the previous model where the complexity of the formulas was hidden in the Exactly and AtLeast macro operators.

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

3.3 Strategies

TODO: Nadefinovat strategii obecněji - v závislosti na historii atd. Potom definovat třídy strategií - memory-less pokud se rozhodují jen podle formule; non-adaptive pokud se rozhodují jen podle čísla experimentu a na formule nekoukaj

Definition 11. A *strategy* is a function $\sigma : \mathrm{Form}_X \to E$, determining the next experiment for given accumulated knowledge. (Note that this requires)

A strategy σ together with a valuation v (satisfying φ_0) induce a *solving process*, which is an infinite sequence

$$\lambda_{\sigma,v} = \varphi_0 \xrightarrow{e_1} \varphi_1 \xrightarrow{e_2} \varphi_2 \xrightarrow{e_3} \dots,$$

where $e_i = \sigma(\varphi_0 \wedge \varphi_1 \wedge \dots \wedge \varphi_{i-1})$ and $\varphi_i \in \Phi(e_i)$ and $v(\varphi_i) = 1$, for all $i \in \mathbb{N}$. Notice that due to the condition in Definition 6, there is always exactly one such ψ_i.
For the sake of simplicity, let us write $\varphi_{0..k}$ instead of $\varphi_0 \wedge \varphi_1 \wedge \dots \wedge \varphi_k$.
We define *length* of the solving process, denoted $|\lambda_{\sigma,v}|$ (despite the infinite length of the sequence), as the smallest $k \in \mathbb{N}_0$ such that $\varphi_{0..k}$ has only one model ($\#_X \varphi_{0..k} = 1$). This corresponds to the point where we can uniquely determine the code. Note that it always holds $\#\varphi_{0..k} > 0$ because $v(\varphi_i) = 1$ for all $i \in \mathbb{N}_0$.

The following lemma is a straightforward consequence of the memory-less nature of the games. It says that once a strategy gives us an experiment that yields no new information, we will never get any new information (using the strategy).

Lemma 12. *If* $\#\varphi_{0..k} = \#\varphi_{0..k+1}$ *for some* $k \in \mathbb{N}$*, then* $\#\varphi_{0..k} = \#\varphi_{0..k+l}$ *for any* $l \in \mathbb{N}$*.*

Proof. If $\varphi_{0..k+1} = \varphi_{0..k} \wedge \varphi_{k+1}$ is satisfied by valuation v, so must be $\varphi_{0..k}$. Since $\#\varphi_{0..k} = \#\varphi_{0..k+1}$, the sets of valuations satisfying $\varphi_{0..k}$ and $\varphi_{0..k+1}$ must be exactly the same and the formulas are thus equivalent. This implies $\sigma(\varphi_{0..k}) = \sigma(\varphi_{0..k+1})$ and thus also $\varphi_{k+2} = \varphi_{k+1}$. By induction, $\varphi_{k+l} = \varphi_{k+1}$ and $\varphi_{0..k+l} \equiv \varphi_{0..k}$ for any $l \in \mathbb{N}$. ∎

TODO: Co nějaká average délka?

The *worst-case number of experiments* Λ^σ of a strategy σ is the maximal length of the solving process $\lambda_{\sigma,v}$ over all models v of φ_0, i.e. $\Lambda^\sigma = \max_{v \in V_X, v(\varphi_0)=1} |\lambda_{\sigma,v}|$. We say that a strategy σ *solves the game* if Λ^σ is finite. The game is *solvable* if there exists a strategy that solves the game.

Definition 13. A strategy σ is *optimal* if $\Lambda^\sigma \leq \Lambda^{\sigma'}$ for any strategy σ'.

Definition 14. Let $f : \mathrm{Form}_X \to \mathbb{Z}$. A strategy σ is *f-greedy* if for every $\varphi \in \mathrm{Form}_X$ and $e' \in E$,

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

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.

3.4 Symmetry Breaking

TODO: Tohle hrozně nefunguje. Musí být symetrické i experimenty. Plán:

- zadefinovat symetrii množiny experimentů jako gropu permutací proměnných tak, aby $\forall e \in E \exists e^\pi$ (kde).

- zadefinovat nějakou vlastnost hry (symetrická) - pokud pro nějaké e existuje e^π, pak pro všechny
- říct, že pokud jsou parametrizace nějak normálně omezené (typicky libovolný řetězec, nebo řetězec bez opakování), tak je ta hra symetrická
- omezit se s analýzou jen na symetrické hry
- zadefinovat konzistentní strategii, jako strategii, která na zpermutovanou formuli zahraje zpermutovaný experiment
- zadefinovat ekvivalenci experimentů podle definice výše
- dokázat, že pokud mám strategie $\sigma_1.\sigma_2$ tak že $\sigma_1(\varphi) \sim_\varphi \sigma_2(\varphi)$, tak existuje permutace proměnných taková, že $|\lambda_{\sigma_1,v}| = |\lambda_{\sigma_2,v^\pi}|$

Definition 15. For an experiment e and a permutation $\pi \in \mathrm{Perm}_X$, a π-symmetric experiment $e^\pi \in E$ is an experiment such that $\{\varphi^\pi \in \Phi(e)\} = \{\varphi \in \Phi(e^\pi)\}$. Clearly, no experiment satisfying this may exists.

Definition 16. A game $\mathcal{G}$ is *symmetric* if it satifies the following implication: If for an experiment e exists a π-symmetric experiment $e^\pi \in E$, then a π-symmetric experiment exists for every experiment.

For the rest of this chapter, the game we analyze is *symmetric*.

Definition 17. A memory-less strategy σ is *consistent* if and only if $\sigma(\varphi^\pi) = \sigma(\varphi)^\pi$ for any $\varphi \in \mathrm{Form}_X$ and $\pi \in$ symmetry group of the set of experiments.

Definition 18. An experiment $e_1 \in E$ is equivalent to $e_2 \in E$ with respect to φ, written $e_1 \sim_\varphi e_2$, if and only if there exists a permutation $\pi \in \mathrm{Perm}_X$ such that $\{\varphi \wedge \psi \mid \psi \in \Phi(e_1)\} \equiv \{(\varphi \wedge \psi)^\pi \mid \psi \in \Phi(e_2)\}$.

Theorem 19. *Let σ_1, σ_2 be two consistent strategies, such that $\sigma_1(\varphi) \sim_\varphi \sigma_2(\varphi)$ for any $\varphi \in Form_X$. Then ... ?!?*

Proof. Značme $\varphi_k, \chi_{k,\pi}$ accumulated knowledge of σ, τ after k steps on valuation v, v^π respectively.
Indukcí ukážu, že pro $k \in \mathbb{N}_0$ existuje $\pi \in Sym(E)$, $\varphi_k^\pi = \chi_{k,\pi}$. Pro $k = 0$ zřejmě platí. TODO: Sym(E) vs symetrie φ_0..
Předpokl. $\varphi_k^\pi = \chi_{k,\pi}$. Označme $e_1 = \sigma(\varphi_k)$, $e_2 = \tau(\chi_{k,\pi})$.

$$e_2 = \tau(\chi_{k,\pi}) \sim_{\chi_{k,\pi}} \sigma(\chi_{k,\pi}) = \sigma(\varphi_k^\pi) = \sigma(\varphi_k)^\pi = e_1^\pi$$

Díky tomu existuje $\rho \in Sym(E)$ such that

$$\begin{aligned}\{\chi_{k,\pi} \wedge \psi \mid \psi \in \Phi(e_2)\} &= \{(\chi_{k,\pi} \wedge \psi)^\rho \mid \psi \in \Phi(e_1^\pi)\} = \\ &= \{(\varphi_k^\pi \wedge \psi^\pi)^\rho \mid \psi \in \Phi(e_1)\} = \{(\varphi_k \wedge \psi)^{\pi\rho} \mid \psi \in \Phi(e_1)\}\end{aligned}$$

$\rho \in Sym(E)$takže i $\pi\rho \in Sym(E)$. Ukážeme, že $\varphi_{k+1}^{\pi\rho} = \chi_{k+1,\pi\rho}$.

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

5 Implementation details

5.1 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.

5.2 Sat solving

6 Conclusions

Bibliography