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 Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.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

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

Definition 5 (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 $\circ \in \{\wedge, \vee, =>\}$. 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 $f(\$n)$ apperas in ψ 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, 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, 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).

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. 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 $\#\lambda\langle|\lambda|\rangle = 1$, 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 $e \in E$,

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

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

  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 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 $1 \leq i \leq n$,

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

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.
- $T = \{g_{k_1,\ldots,k_m} \mid k_i \in \{1, \ldots, n\}, \sum_i k_i = n\}$,
  $\Sigma = C$,
  $E = \{(g_{k_1,\ldots,k_m}, p) \mid p \in \Sigma^n, \#i.(p[i] = j) = k_j\}$.
  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 $1 \leq i \leq n$,

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

TODO: Zdůvodnit proč to funguje.

Example 11 (Mastermind (alternative)). $\mathcal{M}'_{n,m} = (X, \varphi_0, T, \Sigma, E, F, \Phi)$.

- X and φ_0 is defined as in Example 10.

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

- $F = \{f_1, \ldots, f_n\}$, where $f_i(c) = x_{i,c}$ for $1 \le i \le n$, $\Phi(g) = \{\text{Outcome}(b,w) \mid 0 \le b \le n, 0 \le w \le n, b+w \le n\}$, 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 $= 4$:
...
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

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 (satisfying φ_0) 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.

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

This corresponds to the point where we can uniquely determine the code.

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.

$$\Lambda^\sigma = \max_{v \in \mathtt{Val}_X,\, v(\varphi_0)=1} |\sigma|_v.$$

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.

The *average-case number of experiments* $\Lambda^\sigma_{\mathrm{exp}}$ 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.

$$\Lambda^\sigma_{\mathrm{exp}} = \frac{\sum_{v \in \mathtt{Val}_X,\, v(\varphi_0)=1} |\sigma|_v}{\#\varphi_0}.$$

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 $\Lambda^\sigma_{\mathrm{exp}} \leq \Lambda^{\sigma'}_{\mathrm{exp}}$ 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. ∎

Non-adaptive strategies

Definition 15 (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 16 (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 $\lambda_1\langle|\lambda_1|\rangle \equiv \lambda_2\langle|\lambda_2|\rangle$ then $\sigma(\lambda_1) = \sigma(\lambda_2)$.
Memory-less strategies can be considered as functions $\mathtt{Form}_X \to E$. Then $\sigma(\lambda) = \tau(\lambda\langle|\lambda|\rangle)$

Lemma 17. *Let σ be a memory-less strategy and v a model of φ_0. If there exists $k \in \mathbb{N}$ such that $\#\lambda^\sigma_v\langle k \rangle = \#\lambda^\sigma_v\langle k+1 \rangle$, then $\#\lambda^\sigma_v\langle k \rangle = \#\lambda^\sigma_v\langle k+l \rangle$ for any $l \in \mathbb{N}$.*

Proof. For the sake of simplicity, let $\alpha^k = \lambda^\sigma_v\langle k \rangle$. 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 18. *Let σ be a strategy. Then there exists a memory-less strategy τ such that $|\sigma|_v \geq |\tau|_v$ for all v satisfying φ_0.*

Proof. TODO: Napsat důkaz.

Definition 19 (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 $e' \in E$,

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

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.

Lemma 20. *Let $b = \max_{t \in T} |\Phi(t)|$ be the maximal number of possible outcomes of an experiment. If for any TODO: reachable $\varphi \in \mathtt{Form}_X$,*

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

then a greedy strategy σ is optimal and

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

Proof. TODO: Napsat důkaz.

Example 21. Greedy strategies are optimal in the fake-coin game $\mathcal{F}_n$. TODO: Napsat důkaz.

3.4 Symmetries in Code Braking Games

Definition 22 (Symmetric experiment). For an experiment $e = (t, p)$ and a permutation $\pi \in \mathtt{Perm}_X$, 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 23 (Symmetry group). We define a *symmetry group* Π as the maximal subset of $\mathtt{Perm}_X$ such that for every $\pi \in \Pi$ and for every experiment $e \in E$, there exists a π-symmetric experiment e^π.

Definition 24 (Consistent strategy). A memory-less strategy σ is *consistent* if and only if for every $\varphi \in \mathtt{Form}_X$ 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 φ a dostanu, že to má dávat různé věci.

Lemma 25. *Let τ be a memory-less strategy. There exists a consistent memory-less strategy σ such that $|\tau|_v \geq |\sigma|_v$ for all $v \in \mathtt{Val}_X$ satisfying φ_0.*

Proof. TODO: Idea: když chci nadefinovat $\sigma(\varphi)$, kouknu na všechny v, k, takové, že $\exists \pi \in \Pi(\varphi^\pi = \lambda^\tau_v\langle k\rangle)$, pro každý si spočítám $|\tau|_v - k$ (tj. počet experimentů do konce) a vezmu minimum.

Lemma 26. *Let σ be a consistent memory-less strategy and let v_1, v_2 be two models of φ_0. If v_1 is a model of $\lambda^\sigma_{v_2}\langle k\rangle$, then $\lambda^\sigma_{v_1}\langle i\rangle = \lambda^\sigma_{v_2}\langle i\rangle$ for $i \in \mathbb{N}_0$, $i \leq k$.*

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 accrued knowledge: $\lambda_1(i) = \sigma(\lambda_1\langle i-1\rangle) = \sigma(\lambda_2\langle i-1\rangle) = \lambda_2(i)$. 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. ∎

Definition 27 (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 28. *Let σ, τ be two consistent memory-less strategies, such that $\sigma(\varphi) \cong_\varphi \tau(\varphi)$ for any $\varphi \in \mathtt{Form}_X$. There is a bijection $f : \mathtt{Val}_X \to \mathtt{Val}_X$ 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 $(\lambda^\sigma_v\langle k\rangle)^\pi = \lambda^\tau_{v^\pi}\langle k\rangle$. TODO: Později pak možná potřebuju, že to platí každý $i \leq k$.. For better readability, let $\alpha_k = \lambda^\sigma_v\langle k\rangle$ and $\beta_{k,\pi} = \lambda^\tau_{v^\pi}\langle k\rangle$

For $k = 0$, take $\pi = \mathtt{id}_X$. Clearly, $\lambda^{\sigma}_{v}\langle 0\rangle = \varphi_0 = \lambda^{\tau}_{v^{\mathtt{id}}}\langle 0\rangle$.
For the induction step, suppose $\alpha^{\pi}_{k} = \beta_{k,\pi}$ and $\pi \in \Pi$. Let $e_1 = \sigma(\alpha_k)$, $e_2 = \tau(\beta_{k,\pi})$. It holds

$$e_2 = \tau(\beta_{k,\pi}) \cong_{\beta_{k,\pi}} \sigma(\beta_{k,\pi}) \overset{IH}{=} \sigma(\alpha^{\pi}_{k}) \overset{cons.}{=} \sigma(\alpha_k)^{\pi} = e_1^{\pi}$$

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

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

As $\rho \in \Pi$ and Π is a permutation group, $\pi\rho \in \Pi$.
Since the game is well-formed, v satisfies exactly one formula in $\{\alpha_k \wedge \psi \mid \psi \in \Phi(e_1)\}$. Therefore $v^{\pi\rho}$ satisfies exactly one formula in $\{(\alpha_k \wedge \psi)^{\pi\rho} \mid \psi \in \Phi(e_1)\} = \{\beta_{k,\pi} \wedge \psi \mid \psi \in \Phi(e_2)\}$, which means that $v^{\pi\rho}$ satisfies $\beta_{k,\pi}$. From Lemma 26, $\beta_{k,\pi} = \beta_{k,\pi\rho}$. Both $\alpha^{\pi\rho}_{k+1}$ and $\beta_{k+1,\pi\rho}$ is thus the only formula from (*) satisfied by $v^{\pi\rho}$ and the induction is complete.

■

Corollary 29. *Let σ_1, σ_2 be two consistent memory-less strategies, such that $\sigma_1(\varphi) \cong_{\varphi} \sigma_2(\varphi)$ for any $\varphi \in \mathtt{Form}_X$. 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