

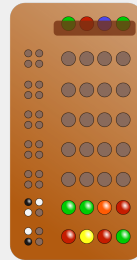
# Algorithmic Analysis of Code-Breaking Games

Mgr. Miroslav Klimoš, prof. RNDr. Antonín Kučera Ph.D.  
Faculty of Informatics, Masaryk University, Brno



## Code-Breaking Games

- ▶ 2 players: *codemaker* and *codebreaker*
- ▶ Codemaker selects a *secret code*
- ▶ Codebreaker strives to reveal the code through a series of *experiments* whose outcomes give partial information about the code
- ▶ Example: Mastermind
  - ▷ Secret code: combination of  $n$  coloured pegs
  - ▷ Codebreaker makes guesses (experiments)
  - ▷ Guesses are evaluated with *black and white markers*
  - ▷ Black marker = correct both colour and position
  - ▷ White marker = the colour is present at a different position
- ▶ Example: Counterfeit Coin
  - ▷ Problem of finding an odd-weight coin using balance scale
  - ▷ Secret code: identity of the unique counterfeit coin
  - ▷ Codebreaker puts coins on the balance scale and observes the outcome



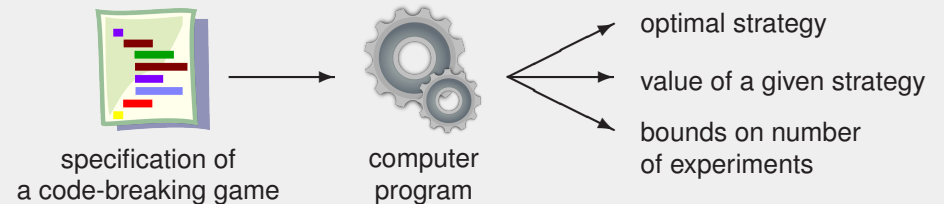
## Questions and Problems

*How should the codebreaker play in order to minimize the number of experiments needed to undoubtedly determine the code?*

*Is there a strategy for experiment selection that guarantees revealing the code after at most  $k$  experiments?*

*What strategy is optimal with respect to the average-case number of experiments, given that the code is selected from the given set with uniform distribution?*

- ▶ We create a computer program to automatically answer these questions



## Steps Towards Automatic Analysis

### 1. Creating a general, formal model of code-breaking games

- ▷ Model based on propositional logic
- ▷ Secret code = valuation of variables
- ▷ Partial information = logical formula

$$\Phi_t = \{ (f_x(\$1) \wedge \neg y) \vee (f_x(\$2) \wedge y), \\ (f_x(\$1) \wedge y) \vee (f_x(\$2) \wedge \neg y), \\ \neg f_x(\$1) \wedge \neg f_x(\$2) \}.$$

### 2. Proposing general strategies for experiment selection

- ▷ "Select an experiment that minimizes the maximal number of possibilities for the code in the next round"
- ▷ Several strategies of this kind formalized within the model

$$f(\Psi) = \frac{\sum_{\varphi \in \Psi} (\#\varphi)^2}{\sum_{\varphi \in \Psi} \#\varphi}$$

### 3. Developing algorithms for strategy evaluation and synthesis

- ▷ Based on intelligent backtracking
- ▷ Symmetry detection reduces the size of the state-space



### 4. Designing a computer language for game specification

- ▷ Corresponds to the formal model
- ▷ Built on top of Python for easier generation

```
for m in range(1, N//2 + 1):  
    EXPERIMENT("weighing" + str(m), 2*m)  
    OUTCOME("lighter", "(!s) & !y) ...  
    OUTCOME("heavier", "(!s) & y) ...
```

### 5. Implementing proposed algorithms in a computer program

- ▷ Command-line tool written in C++
- ▷ Use of modern SAT solvers for satisfiability queries needed by the algorithms

### 6. Using the program to create new and reproduce existing results

- ▷ Easy reproduction of some of the existing results for Mastermind
- ▷ Automatic analysis of generalizations and other code-breaking games

