Puzzles for Learning Model Checking, Model Checking for Programming Puzzles, Puzzles for Testing Model Checkers [1]

N.V. Shilov[2] K. Yi

Research On Program Analysis System (ROPAS)
Department of Computer Science
Korean Advanced Institute of Science and Technology (KAIST)
Kusong-dong Yusong-gu 373-1
Taejon 305-701, Republic of Korea
`{shilov, kwang}@ropas.kaist.ac.kr`
`http://ropas.kaist.ac.kr/{~shilov, ~kwang}`

Abstract

Paper discuss some issues related to model checking: (1) early model checking teaching via games, (2) utility of model checking and games for solving puzzles, (3) importance of games for validation of model checkers. Basic ideas are illustrated by a model checking based solution for a complicated puzzle how to identify a unique false coin among given ones balancing them limited times.

1 Introduction

The role of formal methods in the development of computer hard- and software increases since systems become more complex and require more efforts for their specification, design, implementation and verification. Simultaneously formal methods become more complicated since they have to capture real properties of real systems for sound reasoning. The best way to get opinion about scope and range of formal methods research and their industrial-strength applications is to visit special sites

http://archive.comlab.ox.ac.uk/formal-methods.html in Oxford,
http://shemesh.larc.nasa.gov/fm/ in NASA

or from proceedings of the latest World Congress on formal methods FM'99 [1].

A survey of formal methods is out of the scope of this paper. Nevertheless let us remark that specification languages which are in use in formal methods range from propositional to high-order level while a proving technique is either semantical (model-checking) or syntactic (deduction) reasoning. In particular program logics are modal logics used in hard- and software verification and specification. A special place in a diversity of *propositional* program logics belongs

[1]This work is supported by Creative Research Initiatives of the Korean Ministry of Science and Technology

[2]While on leave from A.P. Ershov Institute of Informatics Systems of Siberian Division of Russian Academy of Science, Novosibirsk, Russia

to the propositional μ-Calculus (μC) of D. Kozen [3] due to its expressiveness. In brief μC can be defined as a polymodal variant of basic modal logic **K** with fixpoints. A model checking problem for the μ-Calculus is a very important research topic [7, 8, 4, 5, 6, 12, 13, 10, 11, 14, 9]. Close relations between model checking the μ-Calculus and special bisimulation games are under investigation in papers [12, 13, 14]. In particular, [12] defined *infinite* model checking games. Then [13] defined *finite* fixed point games and characterized indistinguishability of processes (states) by means of formulae with bounded amounts of modalities and fixpoints in terms of winning strategies with bounded amounts of moves. The last cited paper [14] exploited model-checking games for practical efficient local model checking. In contrast, we would like to discuss three other issues related to a role of games for the μ-Calculus, namely:

- finite games for early teaching of model checking (sections 2, 3),
- model checking and abstraction for programming puzzles (sections 4, 5),
- validation of model checkers via game test-suits (section 6, 7).

2 Learning μ-Calculus via games

In spite of the importance of the formal approach for development of a reliable hard- and software this research domain is not well acquainted to non-professionals. In particular, many undergraduate students of departments which are closely related to further progress of computer hard- and software (i.e. pure/applied mathematics and electric/electronic engineering) consider formal methods in general to be out of scope of their interests, since they (formal methods) are

- either too poor for their pure mathematics,
- either too pure for their poor mathematics.

We are especially concerned by this disappointing not well motivated attitude and suppose that a deficit of popular lectures, tutorials and papers on this topic is the main reason for this ignorance. But an attitude of researchers engaged with formal methods to popularization of their research domain seems not to be very much better then an attitude of students to formal methods, while popular presentation of research is a good opportunity for better computer science education and will support the transfer of scientific research to industrial practice. Really, in a comparison with an attitude of mathematicians to popularization of their researches a corresponding activity of computer scientists seem non-impressive. There are several popular world-wide known journals on mathematics primary oriented to a wide community which comprises students as well as researchers (ex., *The Mathematical Intelligencer* and *The American*

Mathematical Monthly). Moreover, mathematicians become more concerned by popularity of the applied mathematics among students. In particular, SIAM recently launched a special section on education in the SIAM Review. As is stated in the Guidelines for SIREV Authors [15], "In the large majority of cases, articles should be written for students, not to faculty. Articles should provide descriptions, illustrations and sights regarding established or recent knowledge, as opposed to new research results". In contrast it is very hard to point out a journal or magazine in computer science which regularly publishes popular and expository papers primary oriented for students.

The above arguments can lead to a conclusion that an attitude to popular presentation of formal methods theory is negative. This conclusion is invalid of course. Really, let us just to remind brilliant lectures of professors E. W. Dijkstra and D. Gries, International Summer Schools in Marktoberdorf (http://www4.in.tum.de/div/summerschool/) or European Summer Schools in Logic, Language and Information (http://www.cs.bham.ac.uk/ esslli/). But let us also to remark that an auditory of these lectures and school is comparatively small (a couple of hundreds per year) and consists of graduate or postgraduate students, junior scientists or even professors.

Earlier and better learning formal methods via popular (but sound) presentation of mathematical foundations of formal methods can be based on games and game-based puzzles. An educational role of games and game-based puzzles is well acknowledged in the literature on logics of knowledge in computer science. For example, in [2] a knowledge-based analysis of muddy children puzzle, synchronous attack and Byzantine agreement motivate and illustrate basic theoretical ideas and concepts. May be the main lesson which educators/researchers should learn from [2] is: for being attractive mathematical foundations of formal methods should be illustrated by challenging game-based examples.

We would like to sketch below how a program logics tributary creek of a powerful stream called formal methods was presented in a popular (but mathematically sound) form to undergraduate students of mathematics, physics and technical departments of Novosibirsk State University during preparation to the 1999/2000 Regional ACM Collegiate Programming Contest [16]. We hope that this example can be interesting for model checking community not only from a pedagogical viewpoint since it assists to understand and develop some further results presented in the paper.

The first author of this paper was a member of the program committee for a regional middle school contest on mathematics. Several years ago the following puzzle was suggested by the committee to pupils on the contest:

• A set of coins consists of 14 valid and 1 false coin. All valid coins have one and the same weight while the false coin has a different weight. One of the valid coins is marked while all other coins (including the false one) are unmarked. Is it possible to identify a false coin balancing coins 3 times atmost?

Both authors were trainers of teams of undergraduates for ACM Regional programming contests. So a question how to put the puzzle for programming arose naturally. Finally a corresponding programming problem was designed and offered to undergraduate students on training sessions. A brief form of the problem follows:
• Write a program with 3 inputs
- a number N of coins under question,
- a number M of marked valid coins,
- a limit of balancing K
which outputs either `impossible` or another executable interactive program `ALPHA` (in the same language) with respect to existence of a strategy to identify a unique false coin among N coins with use of M marked valid coins and balancing coins K times at most. Each session with `ALPHA` should begin from user's choice of a number of a false coin and weather it is lighter or heavier. Then a session consists of a series of rounds and general amount of rounds in the session can not exceed K. On each round the program outputs two disjoint subsets of numbers of coins to be placed on pans of a balance. The user in his/her turn replies in accordance with his initial choice. The session finishes with the the final output of the program `ALPHA` – a number of the false coin.

Here the programming problem is over. Since the problem is to write a program which produce another program then we would like to refer to the first program as the *metaprogram* and to the problem as the *metaprogram problem* respectively. For tackle the problem let us give below a game interpretation.
• Let M and N be non-negative integer parameters and let $(N + M)$ coins be enumerated by consequent numbers from 1 to $(N + M)$. Coins with numbers in $[1..M]$ are valid while there is a unique false among coins with numbers in $[(M + 1)..(M + N)]$. The *GAME(N,M)* of two players *user* and *prog* consists of a series of rounds. On each round a move of the *prog* is a pair of disjoint subsets (with equal cardinalities) of $[1..(M + N)]$. A possible move of the *user* is either $<$, $=$ or $>$, but a move must be *consistent* with all constraints induced in previous rounds. *Prog* wins the *GAME(N,M)* as soon as a *unique* number in $[1..(M + N)]$ satisfies all constraints induced during the game.

In these settings the metaprogram problem can be reformulated as follows:
• Write a program which for all $N \geq 1$, $K \geq 0$ and $M \geq 0$ generates (iff possible) a K-rounds at most wining strategy for *prog* in the *GAME(N,M)*.

Then another question arises: What is a proper formalism for solving the metaprogram problem in this new setting? A hint is a formalism should have explicit fixpoints. Really, the following informal statement is very natural: If a player *prog* is in a position where he/she has a winning strategy then he/she has a move prior to and after which the game is not lost, but after which every move of another player *user* leads to a position where the game is lost or *prog* has a winning strategy. In other words: a set of positions where *prog* has a winning strategy is a fixpoint of a special mapping from sets of position to sets of positions.

A functional paradigm is a well known paradigm with explicit fixpoints. But, in accordance with regulations of ACM Collegiate Programming Contests, programs should be written in an imperative language PASCAL, C or C++. So another paradigm would be better in this case then the functional one. It should be a paradigm which captures imperative style and fixpoints simultaneously. This is a program logics paradigm in general, and the formalism of the propositional μ-Calculus in particular. But this formalism is not very simple since in the most comprehensive form it relies upon transfinite induction: it is not easy to make the μ-Calculus easy for the undergraduate students! Another hint is an incremental approach to the introduction of the μ-Calculus and a concentration on model checking of the μ-Calculus in finite models only. See [17] for full story with a solution for the metaprogram problem and details of background theory while a brief summary is presented in the next sections 3, 4, 5.

3 Program Logics via Games

Let $\{true, false\}$ be boolean constants, *Prp* and *Act* be disjoint finite alphabets of *propositional* and *action* variable respectively. The syntax of the classical propositional logic consists of *formulae* and is constructed from propositional variables and boolean connectives $\neg$ (*negation*), $\wedge$ (*conjunction*) and $\vee$ (*disjunction*) in accordance to standard rules. *Elementary Propositional Dynamic Logic* (EPDL)[18] has additional features for constructing formulae — modalities which are associated with action variables: if a is an action variable and ϕ is a formula then $([a]\phi)$ and $(\langle a\rangle\phi)$ are formulae[3]. The semantics of EPDL is defined in models, which are called *Transition Systems* or *Kripke Structures*. A model M is a pair (D_M, I_M) where the *domain* D_M is a nonempty set, while the *interpretation* I_M is a pair of special mappings (P_M, R_M). Elements of the domain D_M are called *states*. The interpretation maps propositional variables into sets of states and action variables into binary relations on states:

$$P_M : Prp \to \mathcal{P}(D_M) \quad , \quad R_M : Act \to \mathcal{P}(D_M \times D_M)$$

where $\mathcal{P}$ is a power-set operation. We write $I_M(p)$ and $I_M(a)$ instead of $P_M(p)$ and $R_M(a)$ whenever it is implicit that p and a are propositional and action variables. Models can be considered as labeled graphs with nodes and edges marked by sets of propositional and action variables respectively. For every model $M = (D_M, I_M)$ a *validity* relation $\models_M$ between states and formulae can be defined inductively with respect to the structure of formulae. For boolean

[3] which are read as "*box/diamond a ϕ*" or "*after a always/sometimes ϕ*" respectively

constants, propositional variables and propositional connectives semantics is defined in a standard way while we have

1. $s \models_M (\langle a\rangle\phi)$ iff $(s, s') \in I_M(a)$ and $s' \models_M \phi$ for same state s',
2. $s \models_M ([a]\phi)$ iff $(s, s') \in I_M(a)$ implies $s' \models_M \phi$ for every state s'.

So, an experienced mathematician can remark that EPDL is just a polymodal variant a classical basic modal logic **K** [19].

Finite games can illustrate all EPDL-related notions. A *finite game of two plays* A and B is tuple $(P,\ M_A,\ M_B,\ F)$ where

- P is a nonempty finite set of *positions*,
- $M_A, M_B \subseteq P \times P$ are *(possible) moves* of A and B,
- $F \subseteq P$ is a set of *final positions.*

A *session* of the game is a sequence of positions $s_0,\ ...s_n,\ ...$ where all even pairs are moves of one player (ex., all $(s_{2i},\ s_{2i+1})\ \in M_A$) while all odd pairs are moves of another player (ex., all $(s_{2i+1},\ s_{2i+2})\ \in M_B$). A pair of consequentive moves of two players in a session comprises three consequentive positions (ex., $(s_{2i}, s_{2i+1}, s_{2(i+1)})$ is called a *round.* A player *loses* a session iff after a move of the player the session enters a final position for the first time. A player *wins* a session iff another player loses the session. A *strategy* of a player is a subset of the player's possible moves. A *winning strategy* for a player is a strategy of the player which always leads to the player's win: the player wins every session which he/she begins and in which he/she implement this strategy instead of all possible moves. Every finite games (P, M_A, M_B, F) of the above kind can be presented as a finite Kripke structure $G_{(P,M_A,M_B,F)}$ in a natural way:

- states are positions P,
- action variables $move_A$ and $move_B$ are interpreted as M_A and M_B,
- a propositional variable $fail$ is interpreted as F.

Proposition 1 *Let (P, M_A, M_B, F) be a finite game of two players, a formula WIN_0 be $false$ and for every $i \geq 1$ let WIN_{i+1} be a formula $\neg fail \wedge <move_A> \big(\neg fail \wedge [move_B](fail \vee WIN_i)\big)$.*

- *For every $i \geq 0$ the formula WIN_i is valid in those states of $G_{(P,M_A,M_B,F)}$ where a player A has a wining strategy with i-rounds at most.*
- *For every $i > 0$ a first step of every i-rounds at most wining strategy for a player A consists in a move to a position where $\neg fail \wedge [move_B](fail \vee WIN_{i-1})$ is valid.*

Let an infinite disjunction $\bigvee_{i\geq 0} WIN_i$ *with semantics* $\bigcup_{i\geq 1}\{s : s \models_M WIN_i\}$ *in a model* M *be a special extension of EPDL.*

- *An infinite disjunction* $\bigvee_{i\geq 0} WIN_i$ *is valid in those states of* $G_{(P,M_A,M_B,F)}$ *where a player* A *has a wining strategy*
- *An infinite disjunction* $\bigvee_{i\geq 0} WIN_i$ *is illegal formula of EPDL and is not equivalent to any formula of EPDL.*

The above proposition 1 leads to the following suggestion naturally. Let us define the propositional μ-Calculus as an extension of EPDL by two new features: if p is a propositional variable and ϕ is a formula then $(\mu p.\phi)$ and $(\nu p.\phi)$ are formulae[4]. We would like also to impose the following context-sensitive restriction: *No bounded instance of a propositional variable can be negative.* Informally speaking $\mu p.\phi$ is an "abbreviation" for an infinite disjunction

$$false \lor \phi_p(false) \lor \phi_p(\phi_p(false)) \lor \phi_p(\phi_p(\phi_p(false))) \lor ... = \bigvee_{i\geq 0} \phi_p^i(false)$$

while $\nu p.\phi$ is an "abbreviation" for another infinite conjunction

$$true \land \phi_p(true) \land \phi_p(\phi_p(true)) \land \phi_p(\phi_p(\phi_p(true))) \land ... = \bigwedge_{i\geq 0} \phi^i(true),$$

where $\phi_p(\psi)$ is a result of substitution of a formula ψ on places of p in ϕ, $\phi_p^0(\psi)$ is ψ, and $\phi_p^{i+1}(\psi)$ is $\phi_p(\phi_p^i(\psi))$ for $i \geq 0$. In spite of informal character of the above semantics basically the formal one in finite models is just the same. For every finite model $M = (D_M, I_M)$ the *validity* relation $\models_M$ between states and formulae of EPDL can be extended on formulae of the μ-Calculus as follows:

3. $s \models_M (\mu p.\phi)$ iff $s \models_M \phi_p^i(false)$ for some $i \geq 0$;
4. $s \models_M (\nu p.\phi)$ iff $s \models_M \phi_p^i(true)$ for every $i \geq 0$.

In particular, if ϕ is a formula

$$\neg fail \land < move_A > \big(\neg fail \land [move_B](fail \lor win)\big)$$

where win is a propositional variable, then the formula WIN_0 is just $false \equiv \phi_{win}^0(false)$, while WIN_{i+1} $(i \geq 0)$ is

$$\phi_{win}^{i+1}(false) \equiv \neg fail \land < move_A > \big(\neg fail \land [move_B](fail \lor \phi_{win}^i(false))\big).$$

In terms of the μ-Calculus the above proposition 1 can be reformulated:

Proposition 2 *Let* (P, M_A, M_B, F) *be a finite game of two players, a formula* WIN *be* μ *win.*$\Big(\neg fail \land < move_A > \big(\neg fail \land [move_B](fail \lor win)\big)\Big)$.

[4] which are read as "*mu/nu p* ϕ" or "*the least/greatest fixpoint p of* ϕ" respectively

- *For every $i \geq 0$ the formula $WIN^{i}_{win}(false)$ is valid in those states of $G_{(P,M_A,M_B,F)}$ where a player A has a wining strategy with i-rounds at most.*

- *For every $i > 0$ a first step of every i-rounds at most wining strategy for a player A consists in a move to a position where $\neg fail \wedge [move_B](fail \vee WIN^{i-1}_{win}(false))$ is valid.*

- *A formula WIN is valid in those states of $G_{(P,M_A,M_B,F)}$ where a player A has a wining strategy*

- *A formula WIN is not equivalent to any formula of EPDL.*

4 Towards Metaprogram via Model Checking

Now we are ready to return to a parameterized game *GAME(N,M)* ($N \geq 1$, $M \geq 0$) and describe formally a *concrete model* for it which we would like to denote by *GAME(N,M)* also. Positions in this parameterized game are tuples (U, L, H, V, Q) where

- U is a set of coin numbers in $[(M+1)..(M+N)]$ which are currently under question but which *were not* tested against other coins;

- L is a set of coin numbers in $[(M+1)..(M+N)]$ which are currently under question but which *were* tested against other coins and turned to be *lighter*;

- H is a set of coin numbers in $[(M+1)..(M+N)]$ which are currently under question but which *were* tested against other coins and turned to be *heavier*;

- V is a set of coin numbers in $[1..(N+M)]$ which are currently known to be *valid*;

- Q is a current *balancing query*, i.e. a pair of disjoint subsets of $[1..(N+M)]$ of equal cardinality.

Three constraints are natural: (1)U, L, H and V are disjoint, (2)$U \cup L \cup H \cup V = [1..(N+M)]$, (3)$U \cup L \cup H \neq \emptyset$. In addition we can claim that (4)$U \neq \emptyset$ iff $L \cup H = \emptyset$ since a unique false is among untested coins iff all previous balancings gave equal weights, and (5)if $Q = (S_1, S_2)$ then either $S_1 \cap V = \emptyset$ either $S_2 \cap V = \emptyset$ since it is not reasonable to add extra valid coins on both pans of a balance. A possible move of a player *prog* is a query for balancing two sets of coins, i.e. pair of positions

$$(U, L, H, V, (\emptyset, \emptyset)) \xrightarrow{prog} (U, L, H, V, (S_1, S_2))$$

where S_1 and S_2 are disjoint subsets of $[1..(N+M)]$ with equal cardinalities. A possible move of a player *user* is a reply $<$, $=$ or $>$ for a query which causes position change

$$(U, L, H, V, (S_1, S_2)) \xrightarrow{user} (U', L', H', V', (\emptyset, \emptyset))$$

in accordance with the query and a reply $<$, $=$ or $>$. We do not like an idea to present a complicated (while concise) definition for $\xrightarrow{user}$. In contrast we would like to give some intuition behind it. For it let us represent S_1 as $U_1 \cup L_1 \cup H_1 \cup V_1$ and S_2 as $U_2 \cup L_2 \cup H_2 \cup V_2$ respectively, where $U_{1,2} \subseteq U$, $L_{1,2} \subseteq L$, $H_{1,2} \subseteq H$, $V_{1,2} \subseteq V$. Since $U \neq \emptyset$ iff $L \cup H = \emptyset$ then there are two disjoint cases: $U = \emptyset$ XOR $L \cup H = \emptyset$. Let us consider the first one only since the second is similar. In this case $U_1 = U_2 = U' = \emptyset$ and

$$L' = \begin{cases} L_1 \text{ if the reply is } <, \text{ since in this case a false coin is} \\ \quad \text{either in } L_1 \text{ and is lighter either it is in } H_2 \text{ and is heavier;} \\ (L \setminus (L_1 \cup L_2)) \text{ if the reply is } =, \text{ since in this case a false coin is} \\ \quad \text{neither in } L_1 \text{ or } L_2 \text{ neither it is in } H_1 \text{ or } H_2; \\ L_2 \text{ if the reply is } >, \text{ since in this case a false coin is} \\ \quad \text{either in } L_2 \text{ and is lighter either it is in } H_1 \text{ and is heavier;} \end{cases}$$

$$H' = \begin{cases} H_2 \text{ if the reply is } <, \text{ since in this case a false coin is} \\ \quad \text{either in } L_1 \text{ and is lighter either it is in } H_2 \text{ and is heavier;} \\ (H \setminus (H_1 \cup H_2)) \text{ if the reply is } =, \text{ since in this case a false coin is} \\ \quad \text{neither in } L_1 \text{ or } L_2 \text{ neither it is in } H_1 \text{ or } H_2; \\ H_1 \text{ if the reply is } >, \text{ since in this case a false coin is} \\ \quad \text{either in } L_2 \text{ and is lighter either it is in } H_1 \text{ and is heavier.} \end{cases}$$

A final position is a position $(U,\ L,\ H,\ V,\ (\emptyset, \emptyset))$ where $|U| + |L| + |H| = 1$. Here a game and a corresponding concrete model are over.

Model checking [20] is a testing a model against a formula. Let us be more concrete. Models for logics which are under consideration in the paper are Transitions Systems or Kripke Structures where states of a system are presented as nodes of a graph but transitions between states are presented as labeled edges. The *global (model) checking* problem consists in a calculation of *the set of all states* of a model where a formula is valid, while the *local (model) checking* consists in testing the validity of a formula in a state of a model. We are especially interested in model checking problem for finite models. For these models both model checking problems are polynomially equivalent so we would not like to distinguish them for finite models. More important topic are parameters used for measuring complexity. If $M = (D_M, (R_M, P_M))$ is a finite model then let d_M, r_M and m_M be amount of states in D_M, amount of edges in R_M and $(d_M + r_M)$ respectively. If a model M is implicit then we would like to use these parameters without subscripts, i.e. just d, r and m. If ϕ is a formula then let f_ϕ be a size of ϕ as a string. If a formula ϕ is implicit then we would like to use this parameter f_ϕ without subscript, i.e. just f. Best known model checking algorithms for the μ-Calculus and finite models are exponential, ex.,

$$O(m \times f) \times \left(\frac{m \times f}{a}\right)^{a-1}$$

is a time bound of Faster Model Checking Algorithm (FMC-algorithm) [4], where an alternating depth a of a formula is a maximal amount of alternations in nesting μ and ν with respect to the syntactical dependences and formally is defined by induction. We would like to point out only that the alternating depth is always less then or equal to the nesting depth of fixpoints for every formula. In particular, the a complexity of model checking a fixed formula WIN in a model is linear in a size of the model.

In terms of model checking a preliminary high-level design for the metaprogram problem is quite simple:

1. (a) input amounts N and M of coins under question and of valid coins, a total amount of balancing K;
   (b) to model check formulae WIN^i_{win} for all $i \in [0..K]$ in the concrete model $GAME(N, M)$;
   (c) if WIN^K_{win} is valid in an *initial* position $([(M+1)..(M+N)], \emptyset, \emptyset, [1..M], (\emptyset, \emptyset))$ then go to 2 else - output an impossibility of a strategy and halt;
2. output a program, which model check formulae $\neg fail \wedge [move_B](fail \vee WIN^i_{win}(false))$ for $i \in [0..(K-1)]$ in the concrete model $GAME(N, M)$ and has K interactive rounds with its user as follows: for every $i \in [1..K]$ downwards (i.e. from $i = K$ to $i = 1$) it outputs to user a move from a current position to an intermediate position where a formula $\neg fail \wedge [move_B](fail \vee WIN^{i-1}_{win}(false))$ is valid, then inputs user's reply $<$, $=$ or $>$ and defines a next position.

In accordance with proposition 2 this design is correct. Concrete models are quite good from pure mathematical viewpoint and an idea to implement, plug and play the above preliminary design seems to be natural. Sorry, concrete models are too large from viewpoint of computer science since amounts of possible positions and possible moves are an exponential functions of N.

5 Towards Metaprogram via Abstraction

A hint how to solve the metaprogram problem is quite easy: to consider *amounts of coins* instead of *coin numbers*. This idea is natural: when somebody is solving puzzles he/she operates in terms of amounts of coins of different kinds not in terms of their numbers!

Now we are ready to discuss a new *abstract model game(N,M)* for the *GAME(N,M)* ($N \geq 1$, $M \geq 0$). Positions in this parameterized game are tuples (u, l, h, v, q) where

- u is an amount of coins in $[1..N]$ which are currently under question but which *were not* tested against other coins;
- l is an amount of coins in $[1..N]$ which are currently under question but which *were* tested against other coins and turned to be *lighter*;
- h is an amount of coins in $[1..N]$ which are currently under question but which *were* tested against other coins and turned to be *heavier*;
- v is an amount of coins in $[1..(N+M)]$ which are currently known to be *valid*;
- q is a *balancing query*, i.e. a pair of quadruples ((u_1, l_1, h_1, v_1) , (u_2, l_2, h_2, v_2)) of numbers of $[1..(N+M)]$.

Five constraints are closely relate to constraints 1-5 for the $GAME(N,M)$: (1)$u+l+h \leq N$, (2)$u+l+h+v = N+M$, (3)$u+l+h \geq 1$, (4)$u \neq 0$ iff $l+h=0$, (5)$v_1 = 0$ or $v_2 = 0$. Additional but natural constraints should be imposed for queries (since we can borrow coins for weighing from available untested, lighter, heavier and valid ones): (6)$u_1+u_2 \leq u$, (7)$l_1+l_2 \leq l$, (8)$h_1+h_2 \leq h$, (9)$v_1+v_2 \leq v$, (10)$u_1+l_1+h_1+v_1 = u_2+l_2+h_2+v_2$. A possible move of a player *prog* is a query for balancing two sets of coins, i.e. pair of positions

$$(u,l,h,v,((0,0,0,0),(0,0,0,0))) \stackrel{prog}{\longrightarrow} (u,l,h,v,((u_1,l_1,h_1,v_1),(u_2,l_2,h_2,v_2))).$$

A possible move of a player *user* is a reply $<$, $=$ or $>$ for a query which causes position change

$$(u,l,h,v,((u_1,l_1,h_1,v_1),(u_2,l_2,h_2,v_2))) \stackrel{user}{\longrightarrow}$$
$$\stackrel{user}{\longrightarrow} (u',l',h',v',((0,0,0,0),(0,0,0,0)))$$

in accordance with the query and a reply:

$$u' = \begin{cases} 0 \text{ if the reply is } <, \\ (u-(u_1+u_2)) \text{ if the reply is } =, \\ 0 \text{ if the reply is } >, \end{cases}$$

$$l' = \begin{cases} (l_1+u_1) \text{ if the reply is } <, \\ (l-(l_1+l_2)) \text{ if the reply is } =, \\ (l_2+u_2) \text{ if the reply is } >, \end{cases}$$

$$h' = \begin{cases} (h_2+u_2) \text{ if the reply is } <, \\ (h-(h_1+h_2)) \text{ if the reply is } =, \\ (h_1+u_1) \text{ if the reply is } >, \end{cases}$$

$$v' \; = \; ((N+M) - (u' + l' + h')).$$

A final position is a position $(u,\ u,\ h,\ v,\ ((0,0,0,0),(0,0,0,0)))$ where $u+l+h = 1$. Thus a new game and a corresponding abstract model are over. An overall amount of possible position moves in $game(N,M)$ is less then $\frac{(N+1)^6}{6}$.

What is an utility of abstraction? In general, let Φ be a set of formulae, $M_1 = (D_1, I_1)$ and $M_2 = (D_2, I_2)$ be two models, and $g : D_1 \to D_2$ be a mapping. The model M_2 is called an *abstraction* of the model M_1 with respect to formulae Φ iff[5] for all formula $\phi \in \Phi$ and state $s \in D_1$ the following holds: $s \models_1 \phi \Leftrightarrow g(s) \models_2 \phi$. In particular let us consider models $Game(N,M)$ and $game(N,M)$ ($N \geq 1$, $M \geq 0$) and define a *counting* mapping $count \; : \; D_{GAME(N,M)} \to D_{game(N,M)}$ as follows:

$$count \; : \; (U, L, H, V, (S_1, S_2)) \; \mapsto (|U|, |L|, |H|, q)$$

where q is

$$((|S_1 \cap U|, |S_1 \cap L|, |S_1 \cap H|, |S_1 \cap V|), (|S_2 \cap U|, |S_2 \cap L|, |S_2 \cap H|, |S_2 \cap V|)).$$

This counting mapping can be component-withe extended on pairs of positions. The following property of the counting mapping is straightforward:

• For all $N \geq 1$ and $M \geq 0$ the counting mapping is a homomorphism of a labeled graph $GAME(N,M)$ onto another labeled graph $game(N,M)$ with the following property for every position (U,L,H,Q) in the $GAME(N,M)$:

1. *count* maps all moves of a player which begins in the position onto moves of the same player in the $game(N,M)$ which begins in $count(U,L,H,Q)$;

2. *count* maps all moves of a player which finishes in the position onto moves of the same player in the $game(N,M)$ which finishes in $count(U,L,H,Q)$.

It immediately implies the following assertion:

• For all $N \geq 1$ and $M \geq 0$ a model $game(N,M)$ is an abstraction of another model $GAME(N,M)$ with respect to formulae of the μ-Calculus written with use of the unique free propositional variable $fail$ and two action variables $move_A$ and $move_B$ only.

Now we are ready to revise a preliminary high-level design and present an abstraction-based version of a final high-level design for the metaprogram:

1. (a) input numbers N and M of coins under question and of valid coins, a total amount of balancing K;
   (b) to model check formulae WIN^i_{win} for all $i \in [0..K]$ in the abstract model $game(N,M)$;
   (c) if WIN^K_{win} valid in an *initial* position $(N,0,0,M,((0,0,0,0),(0,0,0,0)))$ then go to 2 else - output an impossibility of a strategy and halt;

[5] g is called an *abstraction mapping* in this case.

2. output a program, which model check formulae $\neg fail \wedge [move_B](fail \vee WIN^i_{win}(false))$ for $i \in [0..(K-1)]$ in the abstract model $game(N, M)$ and has K interactive rounds with its user as follows: for every $i \in [1..K]$ downwards (i.e. from $i = K$ to $i = 1$) it outputs to user a move from a *current position* to an *intermediate position* for which a formula $\neg fail \wedge [move_B](fail \vee WIN^{i-1}_{win}(false))$ is valid on $count(intermediate\ position)$ in the abstract model; then it inputs user's reply $<$, $=$ or $>$ and defines a *next position* in accordance with the reply and abstraction mapping $count$.

Since abstract models are abstractions of concrete ones, a correctness of the final high-level design follows from proposition 2. – Implement, plug and play!

6 Testing model checkers via games

An importance of teaching program logics and model checking discussed above in section 2 is based on importance of model checking applications. The main area of a model-checking applications is automatic verification of hard- and software presented as finite state systems [20] while an automatic model checking verification of high-level software specifications [21, 22] or automatic test generation [23] are rapidly developing new application domains. We suppose that in both cases a high-level reliability of model checkers is of extreme importance due to an automatic character of model checking. But in spite of importance of reliability issues of verification tools there are weak moves only in the formal verification community. Let us discus some of reasons of this situation with reliability. First, in automated deduction a reliability problem can most likely be solved by coupling a prover with a proof checker so that the prover will be required to make proofs that can be checked by the proof checker. This approach seems reasonable due to its simplicity and since proofs are relatively short in a comparison with the size of a system to be verified, while proof checking has a linear complexity. Next, the most popular model checkers SMV [24] and SPIN [25] are model checkers for temporal logics, i.e. they use fixpoints on a metalevel only and so that all inner fixpoints are independent of outer ones. In this case model checking algorithms are quite simple and transparent [20].

Unfortunately, both above reasons are invalid for model checkers of the μ-Calculus in finite models. An approach "a la" theorem proving is impossible due to an exponential complexity of model checking "proofs". Simultaneously, a natural transparency of model checking for temporal logics is lost due to a complicated interaction of alternating nesting fixpoints. So we foresee only three reasonable approaches to reliable model checking for the μ-Calculus in finite models:

- simultaneous polyvariant model checking,
- preliminary extensive testing of model checkers,

- formal verification of model checkers.

Due to complexity reasons mentioned above, a polyvariant approach to reliable model checking is time, space and cost expensive. The second approach seems to be problematic also since test-generation is a non-trivial problem itself. This problem is addressed in [26] and is discussed in brief in the next paragraph below. As far as concern formal verification of model checkers then let us point out on a recent paper [27], where an automatically generated from a proof model checker is reported. To the best of our knowledge it is the first and unique paper on formally verified model checkers. This verified model checker is an implementation on Caml of a model checking algorithm from [28], it is generated by an interactive logic framework Coq from a formally presented proof of correctness of the algorithm.

Why an extensive testing of model checkers for the μ-Calculus in finite models is a non-trivial problem? Because overall test suits for a model checker must be transparent, must have predictable results and simultaneously should exploit a non-trivial combinations of fixpoints. But these two claims are mutually exclusive: a predictability of results implies formulae's simplicity, while a non-trivial combinations of fixpoints are non-trivial for forecasting. May be the most appropriate solution for an overall testing of model checkers is to test them against a formally verified model checker on automatically generated test suits.

As far as concerns manual overall testing of model checkers then problem domain of finite games seems to be the best choice for it, since it comprises understandability of formulae and verifiability of results. A correctness of the results in this case can be checked manually or by means of implementing program robots for players simulation. Below we present 3 examples of parameterized finite games which were in use for manual testing of model checkers for the μ-Calculus and finite models in a specification and verification project **REAL** [29, 30, 26, 31, 32]. In the next section 7 we discuss and illustrate another series of examples of special parameterized finite games which are to be implemented in this project for further validation of model checkers. We used parameterized games for tracking how model checkers react on changes of size of models. All examples in this section are clearly presented as a searching problems for winning strategy in finite games of two players while all examples in the next section 7 are presented in a form of puzzles, but we hope that all readers can recognize and formalize the underlying finite games.

The first example is called "Game in Numbers".

- The game is played between two player, Spoiler and Duplicator. Positions are integers between 1 and N or the $fault(\geq N)$. Duplicator starts by picking an initial position. Then Spoiler and Duplicator by turns make moves in accordance to the following rule: from a position i a next position can be i or $i + 10$. Duplicate wins as soon as Spoiler moves to the fault position, Spoiler wins as soon as Duplicator moves to the fault position. Problem: Define all initial positions with a winning strategy for Duplicator.

Another example is called "Millennium Game".

• On the eve of New Year 19NM (N, M $\in$ [0..9]) Alice and Bob played the *millennium game*. Positions in the game were dates of 19NM-2000 years. The *initial position* was a random date from this interval. Then Alice and Bob made moves in their turn: Alice, Bob, Alice, Bob, etc. Available moves were one and the same for both Alice and Bob: if a current position is a *date* then *the next calendar date* and *the same day of the next month* are possible next positions. A player won the game iff his/her counterpart was the first who launched the year 2000. Problem: Define all initial positions with a winning strategy for Alice.

The third example is a metaprogram problem described in the section 2. We would like to remark here that all above examples deal with formulae WIN and $WIN^i_{win}(false)$, $i \geq 0$.

7 More Complicated Test Suits

Examples presented below are more complicated then previous ones since they deal with a more complicated concept of a game with fairness constraints. A correctness of the results in this case can be checked manually or by means of implementing a program robot for a player simulation just as above. Since a role of finite games with fairness constraints is not limited by their utility for overall manual testing of model checkers, we would like to discuss them in details. A *fairness constraint* for a finite game (P, M_A, M_B, F) is a property of positions, i.e. it holds in some positions and does not hold in others. A finite game with fairness constraints is a tuple (P, M_A, M_B, F, C) where (P, M_A, M_B, F) is a finite game, while C is a finite set of fairness constraints. Fairness constrains prohibit sessions where some constraint holds infinitely often: a session is satisfying the constraints C iff every constraint in C holds finitely often only in the session. In contrast, *fairness conditions* prohibit sessions where some condition holds finitely often only. An infinite session is said to be *fair* with respect to a property iff the property holds for an infinite amount of positions in the session. A *winning strategy for sessions which satisfies fairness constraints* for a player is a strategy which guarantee win for the player in every infinite session where constraints hold finitely often only: the player wins every session which he/she begins, which is not prohibited by constraints, and where he/she implements the strategy.

Proposition 3 *Let (P, M_A, M_B, F, C) be a game with fairness constraints while (P, M_A, M_B, F') be another game with the same positions, the same moves but with another final positions F' and without any fairness constraint: F' comprises F and positions where every infinite session is fair with respect to $\vee C$. For every strategy the following is equivalent:*

- *the strategy is a winning strategy for sessions which satisfy fairness constraints in the game* (P, M_A, M_B, F, C)*;*
- *the strategy is a winning strategy in the game* (P, M_A, M_B, F').

Let us consider a formula $\nu q.\big([a]q \wedge \mu r.(p \vee [a]r)\big)$. Where a sub-formula $\phi \equiv \mu r.(p \vee [a]r)$ of this formula is valid in a model? – Just in those states where every infinite a-path eventually leads to p. Really, $\phi_r^0(false) \equiv false$ is always invalid, $\phi_r^1(false) \equiv (p \vee [a]false)$ is valid iff p holds or a does not halts, $\phi_r^2(false) \equiv (p \vee [a]\phi_r^1(false)$ is valid iff p holds or a does not halts after at most 1-step of a,... $\phi_r^{i+1}(false) \equiv (p \vee [a]\phi_r^i(false)$ is valid iff p holds or a does not halts after at most i-steps of a, etc. Where a sub-formula $\nu q.([a]q \wedge \phi)$ of $\nu q.\big([a]q \wedge \mu r.(p \vee [a]r)\big)$ is valid in a model? – Just in those states where every a-path always leads to ϕ. Really, let us consider a formula $\psi \equiv \nu q.([a]q \wedge r)$ and remark that $\psi_r(\phi)$ is $\nu q.([a]q \wedge \phi)$. In this case $\psi_q^0(true) \equiv true$ is always valid, $\psi_q^1(true) \equiv ([a]true \wedge r)$ is equivalent to r, i.e. is valid iff r holds, $\psi_q^2(true) \equiv ([a]\psi_q^1(true) \wedge r)$ is valid iff r holds now and after 1-step of a,... $\psi_q^{i+1}(true) \equiv ([a]\phi_q^i(true) \wedge r)$ is valid iff r always holds during at most i-steps of a, etc. So, in general, $\nu q.([a]q \wedge r)$ is valid iff every a-path always leads to r, and, in particular, $\nu q.([a]q \wedge \phi)$ is valid in those states of a model where every a-path always leads to ϕ. But ϕ itself is valid in those where every infinite a-path eventually leads to p. So $\nu q.([a]q \wedge \phi) \equiv \nu q.\big([a]q \wedge \mu r.(p \vee [a]r)\big)$ is valid in a state of a model iff every infinite a-path infinitely often visits states where p holds. In other words: a formula $\nu q.\big([a]q \wedge \mu r.(p \vee [a]r)\big)$ is valid in a state of a model iff every infinite a-path is fair with respect to p. These arguments and the above proposition 3 imply the following

Proposition 4 *Let* (P, M_A, M_B, F, C) *be a finite game of two players with fairness constraints and* (P, M_A, M_B, F') *be a corresponding finite game without fairness constraints. Let* $FAIR(\vee C)$ *be a formula*

$$\nu q.\Big(([move_A]q \wedge [move_B]q) \wedge \mu r.\big((\vee C) \vee ([move_A]r \wedge [move_B]r)\big)\Big),$$

$FAIL$ *be a formula* $fail \vee FAIR(\vee C)$ *and* $FAIRWIN$ *be another formula*

$$\mu\ win.\Big(\neg FAIL \wedge \langle move_A\rangle\big(\neg FAIL \wedge [move_B](FAIL \vee win)\big)\Big).$$

Then

- $FAIR$ *is valid in those states of the model* $G_{(P,M_A,M_B,F)}$ *where every infinite session is fair with respect to* $\vee C$*;*
- $FAIRWIN$ *is valid in those states of* $G_{(P,M_A,M_B,F)}$ *where the player* A *has a wining strategy in sessions which satisfies the constraints* C.

Let us present an example of a puzzles which can be solved in terms of games with fairness constraints presented above.
• A city consists of squares and roads between them. A taxi driver would like to reach some desirable square (say Rail Way Square) where he/she hopes to get a generous passenger which is ready to pay as much as driver asks. Taxi can move from a square to another square through a road which connect them. Usually driver selects roads in accordance to his/her will, but in some squares (these squares are known) occasional passengers order driver to move through a road in accordance with passenger's choice, which sometimes is a bad, poor road (these roads are known too). But for sake of driver's luck, there is a finite amount of occasional passengers which would like to select these bad roads. Problem: Define from what initial squares driver can reach the desirable square while service all orders of all occasional passengers through a rout?

Let us explain how to represent this puzzle as a finite game with fairness constraints. A hint is to introduce "police stations" in all bad roads. Let positions be all squares and police stations, moves be roads and the desirable square be a final position while a unique fairness constraint be "in a police station". Finally add some additional stops for organizing moves in a proper order (i.e. ...-driver-passenger-driver-...), and the game is ready!

8 Conclusion

The paper presents some non-standard experience with model checking and discusses how regular teaching, training for programming contests and implementation of verification tools can go hand-in-hand with each-other, with model checking and game theory. Simultaneously it is an expression of a strong belief of authors that

- there exists a deficit of popular lectures, tutorials and papers on the topics related to foundations of formal methods;
- challenging programming problems are a good chance for better education and popularization of foundations of formal methods;
- computer science journals and magazines should promote popularization of formal methods foundations.

References

[1] *FM'99 - Formal Methods. World Congress on Formal Methods in the Development of Computer Systems. Toulouse, France, September 1999. Proceedings, Volumes I and II.* Wing J.M., Woodcock J., Davies J. (Eds), Lecture Notes in Computer Science, v. 1708 and 1709, 1999

[2] Fagin R., Halpern J.Y., Moses Y., Vardi M.Y. *Reasoning about Knowledge.* MIT Press, 1995.

[3] Kozen D. *Results on the Propositional Mu-Calculus.* Theoretical Computer Science, v.27, n.3, 1983, p.333-354.

[4] Cleaveland R., Klain M., Steffen B. *Faster Model-Checking for Mu-Calculus.* Lecture Notes in Computer Science, v.663, 1993, p.410-422.

[5] Emerson E.A., Jutla C.S., Sistla A.P. *On model-checking for fragments of Mu-Calculus.* Lecture Notes in Computer Science, v.697, 1993, p.385-396.

[6] Berezine S.A., Shilov N.V. *An approach to effective model-checking of real-time finite-state machines in Mu-Calculus.* Lecture Notes in Computer Science, v.813, 1994, p.47-55.

[7] Cleaveland R. *Tableaux-based model checking in the propositional Mu-Calculus.* Acta Informatica, v.27, 1990, p.725-747.

[8] Bradfield J.C., Stirling S. *Local Model Checking for Infinite State Spaces.* Theoretical Computer Science, v.96, 1992, p.157-174.

[9] Dam M., Fredlund L., Gurov D. *Toward Parametric Verification of Distributed Systems.* Lecture Notes in Computer Science, v.1536, 1998, p.150-185.

[10] Burkart O. *Automatic Verification of Sequential Infinite-State Processes.* Lecture Notes in Computer Science, v.1354, 1997.

[11] Burcart O., Steffen B. *Model-Checking the Full Modal Mu-Calculus for Infinite Sequential Processes.* Lecture Notes in Computer Science, v.1256, 1997, p.419-429.

[12] Stirling C. *Local Model Checking Games.* Lecture Notes in Computer Science, v.962, 1995, p.1-11.

[13] Stirling C. *Local Model Checking Games.* Lecture Notes in Computer Science, v.1055, 1996, p.298-312.

[14] Steven P., Stirling C. *Practical Model Checking Using Games.* Lecture Notes in Computer Science, v.1384, 1998, p.85-101.

[15] *Guidelines for SIAM Review Authors.* http://www.siam.org/journals/sirev/Revguide.htm

[16] *The ACM Collegiate Programming Contest.* http://acm.baylor.edu/acmicpc/default.htm

[17] Shilov N.V., Yi K. *Program Logics Made Easy.* http://ropas.kaist.ac.kr/~shilov/easy.ps

[18] Harel D. *First-Order Dynamic Logic.* Lecture Notes in Computer Science, v.68, 1979.

[19] Bull R.A., Segerberg K. *Basic Modal Logic.* Handbook of Philosophical Logic, v.II, Reidel Publishing Company, 1984 (1-st ed.), Kluwer Academic Publishers, 1994 (2-nd ed.), p.1-88.

[20] Clarke E.M., Grumberg O., Peled D. Model Checking. MIT Press, 1999.

[21] Bultan T., Gerber R., Pugh W. *Model-Checking Concurrent Systems with Unbounded Integer Variables: Symbolic Representation, Approximation, and Experimental Results.* ACM Trans. on Prog. Lang. and Syst., v.21, n.4, 1999, p.747-789.

[22] Anderson R., Beame P., Chan W., Notkin D. *Experiences with the Application of Symbolic Model Checking to the Analysis of Software Specifications.* Lecture Notes in Computer Science, v.1755, 1999, p.???-???.

[23] Gargantini A., Heitmeyer C. *Using Model Checking to Generate Tests from Requirements Specifications.* ACM SIGSOFT Software Engineering Notes, v.24, n.6, 1999, p.146-162 (simultaneously Lecture Notes in Computer Science, v.1687, 1999)

[24] Burch J.R., Clarke E.M., McMillan K.L., Dill D.L., Hwang L.J. *Symbolic Model Checking: 10^{20} states and beyond.* Information and Computation, v.98, n.2, 1992, p.142-170.

[25] Holzmann G.J., Peled D. *The state of spin.* Lecture Notes in Computer Science, v.1102, 1996, p.385-389.

[26] Bodin E.V., Kozura V.E., Shilov N.V. *Experiments with Model Checking for Mu-Calculus in specification and verification project REAL.* Proceedings of the 5-th New Zealand Program Development Colloquium, 1999, p.1-18.

[27] Sprenger C. *A verified Model Checker for the Modal mu-Calculus in Coq.* Lecture Notes in Computer Science, v.1384, p.167-183.

[28] Winskel G. *A note on model checking the modal μ-calculus.* Theoretical Computer Science, v.83, n.2, 1991, p.157-167.

[29] Nepomniaschy V.A., Shilov N.V. *Real92: A combined specification language for systems and properties of real-time communicating processes.* Lecture Notes in Computer Science, v.735, 1993, p.377-393.

[30] Nepomniaschy V.A., Shilov N.V., Bodin E.V. *A concurrent systems specification language based on SDL & CTL.* Proc. of Workshop on Concurrency, Specifications & Programming, Humboldt University, Berlin, Informatik-Bericht Nr.36, 1994, p.15–26.

[31] Nepomniaschy V.A., Shilov N.V., Bodin E.V. *A new language Basic-REAL for specification and verification of distributed system models.* Report Nr. 65 of A.P. Ershov's Institute of Informatics Systems, Novosibirsk, 1999, 39 p. (also available at http://www.iis.nsk.su/edu/publish/bre99/bre99_ps.zip)

[32] Nepomniaschy V.A., Shilov N.V., Bodin E.V. *Specification and Verification of Distributed System by means of Elementary-REAL language.* Programming and Computer Software, v.25, n.4, 1999, p.222-232.