Electronic Notes in Theoretical Computer Science 43 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume43.html 16 pages

Puzzles for Learning Model Checking, Model Checking for Programming Puzzles, Puzzles for Testing Model Checkers

N. V. Shilov [1,2,4] and K. Yi [1,3]

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

Abstract

Paper discusses some issues related to an utility of finite games for early formal methods teaching and for validation of automatic tools which implement formal methods. In particular, some experience with (1) undergraduate teaching model checking via finite games, (2) solving game-based constraints via model checking, (3) testing model checkers against game test suits is presented. 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 validation. At the same time, formal methods become more complicated since they have to capture real properties of real systems for sound reasoning. A survey of formal methods is out of the scope of this paper. The best way to get opinion about scope and range of formal methods research and their industrial-strength applications is to

[1] This work is supported by Creative Research Initiatives of the Korean Ministry of Science and Technology
[2] Email: shilov@ropas.kaist.ac.kr Homepage: http://ropas.kaist.ac.kr/~shilov
[3] Email: kwang@ropas.kaist.ac.kr Homepage: http://ropas.kaist.ac.kr/~kwang
[4] While on leave from A.P. Ershov Institute of Informatics Systems of Siberian Division of Russian Academy of Science, Novosibirsk, Russia

visit special sites `http://archive.comlab.ox.ac.uk/formal-methods.html` in Oxford or `http://shemesh.larc.nasa.gov/fm/` in NASA.

Nevertheless we would like to 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 to the propositional μ-Calculus (μC) of D. Kozen [17] 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 μC is a very important research topic [9,1,10,12,2,24,25,6,7,23,11]. Close relations between model checking the μ-Calculus and special bisimulation games have been studied in papers [24,25,23]. In particular, [24] has defined *infinite* model checking games for μC, [25] has defined *finite* fixed point games and characterized indistinguishability of processes (states) by means of μC formulae in terms of winning strategies, [23] has exploited model-checking games for practical efficient local model checking. 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 for programming winning strategies (sections 4, 5),
- validation of model checkers via finite game test-suits (section 6).

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 for students as well as for researchers (ex., *The American Mathematical Monthly*). Moreover, mathematicians become more concerned by popularity of the applied mathematics among students. In particular, *Society for Industrial and Applied Mathematics* (SIAM) recently launched a special section on education in the *SIAM Review* (SIREV). As is stated in the *Guidelines for SIREV Authors*,

> 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 in computer science which regularly publishes popular and expository papers for students.

The above arguments can mislead to a conclusion that an attitude to popularization of formal methods theory is negative. This conclusion is invalid of course. Really, let us just to remind *International Summer Schools in Marktoberdorf* and *European Summer Schools in Logic, Language and Information.* But let us also to remark that an auditory of these school is comparatively small (a couple of hundreds per year) and consists of graduate or postgraduate students, junior scientists or professors.

Earlier and better teaching 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 [13] 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 [13] 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 for *The ACM Collegiate Programming Contest*. 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.

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 equal weights while the false coin has a another weight. One of the valid coins is marked while all other coins (including the false one) are unmarked. Is it possible to identify the false coin balancing coins 3 times at most?

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 suggested to undergraduate students on training sessions. A brief form of the problem follows:

- Write a program `ALPHA` with 3 inputs
  - a number N of coins under question,
  - a number M of marked valid coins,
  - a limit K of balancing

  which outputs either `impossible` or another interactive program `BETA` (in the same language) which implements a K-times (at most) balancing strategy for identification of a single false coin among N coins with aid of M marked valid coins. Each session with `BETA` should begin from user's choice of a number for a single false coin and weather it is lighter or heavier. Then a session consists of a series of rounds and general amount of rounds 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 the initial choice. The session finishes with the the final output of `BETA` – the number of the false coin.

Since the problem is to write a program which produce another program then we would like to refer the first program as the *metaprogram* and the problem as the *metaprogram problem* respectively.

For tackle the metaprogram problem let us give a game interpretation:

- Let M and N be non-negative integer parameters and let $(N+M)$ coins be enumerated by numbers $[1..(N+M)]$. Coins with numbers $[1..M]$ are valid while there is a single false among coins with a number 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 $[1..(M+N)]$ meets 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 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 operation on sets of position.

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 like PASCAL. 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 [21] for full story with a solution for the metaprogram problem and details of background theory while a brief summary is presented in 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)[15] 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[5]. 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 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 some state s',

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

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 the basic propositional modal logic **K** [3].

Finite games can illustrate all EPDL-related notions. A *finite game of two plays A and B* is a 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 implements 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 3.1 *Let (P, M_A, M_B, F) be a finite game of two players, let WIN_0 be a formula $false$, and for every $i \geq 1$ let WIN_{i+1} be a formula $\neg fail \wedge \langle move_A\rangle(\neg fail \wedge [move_B](fail \vee WIN_i))$.*

- *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 proposition 3.1 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[6]. 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

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

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

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

where $\phi_p(\psi)$ is a result of substitution of ψ instead 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 μC 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

$$\neg fail \land \langle move_A \rangle(\neg fail \land [move_B](fail \lor win))$$

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

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

Thus in terms of μC the proposition 3.1 can be reformulated as follows:

Proposition 3.2 *Let (P, M_A, M_B, F) be a finite game of two players and let WIN be a formula $\mu\ win.\Big(\neg fail \land \langle move_A \rangle(\neg fail \land [move_B](fail \lor win))\Big)$.*

- *For every $i \geq 0$ the formula $WIN_{win}^i(false)$ 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 $\neg fail \land [move_B](fail \lor WIN_{win}^{i-1}(false))$ 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.*

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

4 Towards Metaprogram via Model Checking

Now we are ready to formalize a parameterized game $GAME(N, M)$.

Positions in this 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.

Positions should meet some natural constraints listed below:

- $U \cup L \cup H \neq \emptyset$, since there is a false coin,
- U, L, H and V form a disjoint cover of $[1..(N+M)]$,
- $U \neq \emptyset$ iff $L \cup H = \emptyset$, since
  a false is among untested coins iff all previous balancings gave equal weights,
- 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 final position is a position (U, L, H, V, $(\emptyset, \emptyset)$) with $|U| + |L| + |H| = 1$.

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 would not like to present a complete (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

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

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

Here a game and a corresponding concrete model are done.

Model checking [8] is a testing a model against a formula. 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 an input model where an input formula is valid, while the *local (model) checking* consists in testing the validity of an input formula in an input state of an input 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. For example,

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

is a time bound of Faster Model Checking Algorithm (FMC-algorithm) [10], 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 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 input formula WIN in an input 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:

(i) (a) input integers N, M and K;
  (b) set player A be *prog* and player B be *user*;
  (c) check formula WIN^K_{win} in $GAME(N, M)$;
  (d) if $([(M + 1)..(M + N)], \emptyset, \emptyset, [1..\ M], (\emptyset, \emptyset)) \models_{GAME(N,M)} WIN^K_{win}$, then go to (ii), else - output an impossibility of a strategy and halt;

(ii) output a program, which first check formulae $\neg fail \wedge [move_B](fail \vee WIN^i_{win}(false))$ for $i \in [0..(K - 1)]$ in $GAME(N, M)$ and then has the following loop for $i \in [1..K]$ downwards: output a move to a position where a formula $\neg fail \wedge [move_B](fail \vee WIN^{i-1}_{win}(false))$ holds, input $<$, $=$ or $>$ and define a next position in accordance with it.

In accordance with proposition 3.2 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 exponential functions of N.

5 Towards Metaprogram via Abstraction

Now we are ready to discuss a new *abstract model game(N,M)* for a parameterized $GAME(N, M)$. A hint how to solve the metaprogram problem is quite easy: 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!

Positions in $game(N, M)$ 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 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 final position is a position $(u, l, h, v, ((0, .., 0), (0, .., 0)))$ with $u+l+h = 1$.

A possible move of a player *prog* is a query for balancing two sets of coins, i.e. pair of positions

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

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

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

in accordance with the query and a reply:

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

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

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

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

Thus a new game and a corresponding abstract model are done. An overall amount of possible position moves in $game(N, M)$ is less than $\frac{(N+1)^6}{6}$.

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[7] for every formula $\phi \in \Phi$ and every state $s \in D_1$ the following holds: $s \models_1 \phi \Leftrightarrow g(s) \models_2 \phi$. In particular, let us consider

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

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. It is straightforward that 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 POS in the $GAME(N, M)$: *count* maps all moves of a player which starts/finishes in the position onto moves of the same player in the $game(N, M)$ which starts/finishes in $count(POS)$. It immediately implies that 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 $move_{user}$ and $move_{prog}$ 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:

(i) (a) input integers N, M and K;
   (b) set player A be *prog* and player B be *user*;
   (c) check formula WIN^K_{win} in $game(N, M)$;
   (d) if $(N, 0, 0, M, ((0, .., 0), (0, .., 0)) \models_{game(N,M)} WIN^K_{win}$,
   then go to (ii), else - output an impossibility of a strategy and halt;

(ii) output a program, which first check formulae $\neg fail \wedge [move_B](fail \vee WIN^i_{win}(false))$ for $i \in [0..(K-1)]$ in $game(N, M)$ and then has the following loop for $i \in [1..K]$ downwards: output a move to a position POS such that $count(POS) \models_{game(N,M)} \neg fail \wedge [move_B](fail \vee WIN^{i-1}_{win}(false))$, input $<$, $=$ or $>$ and define a next position in accordance with it.

Correctness of the final high-level design follows from the proposition 3.2, since $game(N, M)$ is an abstraction of $GAME(N, M)$. – 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 [8]. New application domains include verification of high-level software specifications [4] automatic test generation [14], etc. We suppose that in all 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 [5] and SPIN [16] 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 [8].

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 discussed in brief in the next paragraph. As far as a formal verification is concerned, then let us point out on a recent paper [22], 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 [26], 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. 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.

A problem domain of finite games seems to be the best choice for manual overall testing of model checkers, since it comprises understandability of formulae and verifiability of results. A correctness 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** [18,19,20]. We used parameterized games for tracking how model checkers react on model size.

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 $WIN^{i}_{win}(false)$ for $i \geq 0$.

7 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] Bradfield J.C., and S. Stirling *Local Model Checking for Infinite State Spaces*, Theoretical Computer Science, **96** (1992), 157-174.

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

[3] Bull R.A., and K. Segerberg *Basic Modal Logic*, in "Handbook of Philosophical Logic", v.II, Kluwer Academic Publishers, 1994 (2-nd ed.), 1-88.

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

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

[6] Burkart O. "Automatic Verification of Sequential Infinite-State Processes", Lecture Notes in Computer Science **1354**, Springer-Verlag, 1997.

[7] Burkart O., and B. Steffen *Model-Checking the Full Modal Mu-Calculus for Infinite Sequential Processes*, Lecture Notes in Computer Science, **1256** (1997), 419-429.

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

[9] Cleaveland R. *Tableaux-based model checking in the propositional Mu-Calculus*, Acta Informatica, **27** (1990), 725-747.

[10] Cleaveland R., M. Klain, and B. Steffen *Faster Model-Checking for Mu-Calculus*, Lecture Notes in Computer Science, **663** (1993), 410-422.

[11] Dam M., L. Fredlund, and D. Gurov *Toward Parametric Verification of Distributed Systems*, Lecture Notes in Computer Science, **1536** (1998), 150-185.

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

[13] Fagin R., J.Y. Halpern, Y. Moses, and M.Y. Vardi "Reasoning about Knowledge", MIT Press, 1995.

[14] Gargantini A., and G. Heitmeyer *Using Model Checking to Generate Tests from Requirements Specifications*, ACM SIGSOFT Software Engineering Notes, **24** 1999, 146-162

[15] Harel D. "First-Order Dynamic Logic", Lecture Notes in Computer Science, **68**, Springer-Verlag, 1979.

[16] Holzmann G.J., and D. Peled *The state of spin*, Lecture Notes in Computer Science, **1102** (1996), 385-389.

[17] Kozen D. *Results on the Propositional Mu-Calculus*, Theoretical Computer Science, **27** (1983), 333-354.

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

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

[20] Nepomniaschy V.A., N.V. Shilov, and E.V. Bodin *Specification and Verification of Distributed System by means of Elementary-REAL language*, Programming and Computer Software, **25** (1999), 222-232.

[21] N.V. Shilov and K. Yi, *Program Logics Made Easy*, Korea Advanced Institute of Science & Technology, ROPAS Technical Memo No. 2000-7, URL: `http://ropas.kaist.ac.kr/lib/doc/ShYi00.ps.gz`.

[22] Sprenger C. *A verified Model Checker for the Modal mu-Calculus in Coq*, Lecture Notes in Computer Science, **1384** (1998) 167-183.

[23] Steven P., and C. Stirling *Practical Model Checking Using Games*, Lecture Notes in Computer Science, **1384** (1998), 85-101.

[24] Stirling C. *Local Model Checking Games*, Lecture Notes in Computer Science, **962** (1995), 1-11.

[25] Stirling C. *Games and Modal Mu-Calculus*, Lecture Notes in Computer Science, **1055** (1996), 298-312.

[26] Winskel G. *A note on model checking the modal μ-calculus*, Theoretical Computer Science, **83** (1991), 157-167.