Joint NCC & IIS Bull., Comp. Science, 13 (2000), 67–75

Model checking puzzles in μ-Calculus*

N. V. Shilov, K. Yi

Tne paper discusses some issues related to model checking utility and reliability: (1) utility of model checking and games for solving puzzles, and (2) importance of games and puzzles for validation of model checkers.

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. At the same time, 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 the scope and range of research of formal methods 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 [16].

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 to the propositional μ-Calculus (μC) of D. Kozen [21] due to its expressiveness. In brief μC can be defined as a polymodal variant of the basic modal logic **K** with fixpoints. A model checking problem for the μ-Calculus is a very important research topic [12, 3, 13, 15, 4, 32, 33, 9, 10, 31, 14]. Close relations between model checking μC and special bisimulation games are under investigation in papers [32, 33, 31]. In particular, *infinite* model checking games have been defined in [32]. Then, [33] has defined *finite* fixed point games and characterized indistinguishability of 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 [31] has exploited model-checking games for practical efficient local model checking. We would like also to point out that it is very important to express and check existence of a winning strategy in finite games. For example, paper [2] suggests an ability to check winning strategies for american checkers on $n \times n$ desk as a real measure for a power of a model checker. In contrast to papers [32, 33, 31, 2], we would like to discuss two other issues related to the role of games for the μ-Calculus, namely:

- model checking and abstraction for programming puzzles (sections 2, 3),
- validation of model checkers via game test-suits (sections 4, 5).

Importance of puzzles and games for early teaching formal methods is another closely related topic. We would like to remark that (in spite of importance of the formal approach to development of reliable hard- and software) the research domain of formal methods is not well-acquainted to non-professionals. We are especially concerned with disappointing ill-motivated attitude and suppose that a deficit in popular lectures, tutorials and papers on this topic is the main reason for this ignorance (please refer to [27, 28] for detailed discussion). Earlier and better teaching formal methods via popular (but sound) presentation of mathematical foundations of formal methods can be based on games and

*This work is supported by Creative Research Initiatives of the Korean Ministry of Science and Technology

game-based puzzles. The educational role of games and game-based puzzles is acknowledged in the literature on logics of knowledge in computer science. For example, in [17] a knowledge-based analysis of the muddy children puzzle, synchronous attack and Byzantine agreement motivates and illustrates the basic theoretical ideas and concepts. Maybe the main lesson which educators and researchers should learn from [17] is: for being attractive mathematical foundations of formal methods should be illustrated by challenging game-based examples. A similar approach to program logics presentation is exploited in [29] and scketched in [28].

2. Program logics via games

Let $\{true, false\}$ be boolean constants, Prp and Act be disjoint finite alphabets of *propositional* and *action* variables, 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 with standard rules. *Elementary Propositional Dynamic Logic* (EPDL) [19] 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[2]. 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. Semantics of boolean constants, propositional variables and propositional connectives is defined in the standard way:

1. $s \models_M (\langle a\rangle\phi)$ iff $(s, s') \in I_M(a)$ and $s' \models_M \phi$ for some 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 see that EPDL is just a polymodal variant of the classical basic modal logic **K** [6].

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 consecutive moves of two players that comprises three consecutive positions 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 game G of the above kind can be represented as a finite Kripke structure M_G in a natural way:

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

- 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 G be a finite game of two players, a formula WIN_0 be $false$ and for every $i \geq 1$ WIN_{i+1} be a formula $\neg fail \wedge \langle move_A \rangle \big(\neg fail \wedge [move_B](fail \vee WIN_i)\big)$.*

- *For every $i \geq 0$ the formula WIN_i is valid in those states of M_G where a player A has a winning strategy with i-rounds at most.*
- *For every $i > 0$ the first step of every i-rounds at most winning 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 M_G where a player A has a winning strategy.*
- *An infinite disjunction $\bigvee_{i \geq 0} WIN_i$ is an illegal formula of EPDL and is not equivalent to any formula of EPDL.*

The above proposition 1 naturally leads to the following suggestion. 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[3]. 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

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

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

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

where $\phi_p(\psi)$ is a result of substitution of a formula ψ for 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, the formal semantics in finite models is basically 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

$$\neg fail \wedge \langle move_A \rangle \big(\neg fail \wedge [move_B](fail \vee win)\big)$$

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

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

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

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

Proposition 2. *Let G be a finite game of two players, a formula WIN be μ win.* $\Big(\neg fail \wedge \langle move_A\rangle\big(\neg fail \wedge [move_B](fail \vee win)\big)\Big)$.

- *For every $i \geq 0$ the formula $WIN^i_{win}(false)$ is valid in those states of M_G where a player A has a winning strategy with i-rounds at most.*
- *For every $i > 0$ the first step of every i-rounds at most winning 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 M_G where a player A has a winning strategy.*
- *A formula WIN is not equivalent to any formula of EPDL.*

3. Towards metaprogram via model checking

Let us consider the following programming problem:

- 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 the help of M marked valid coins and balancing coins K times at most. Each session with `ALPHA` should begin with the user's choice of the number of a false coin and weather it is lighter or heavier. Then a session consists of a series of rounds and an amount of rounds in the session should not exceed K. At each round the program outputs two disjoint subsets of the numbers of coins to be placed on pans of a balance. The user in turn replies according to his/her initial choice. The session finishes with the final output of the program `ALPHA` — the number of the false coin.

Since the problem is to write a program which produces another program, we would like to refer to the first one as *metaprogram* and to the problem as the *metaprogram problem*. To tackle the problem, let us give 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 *prog* is a pair of disjoint subsets (with equal cardinalities) of $[1..(M + N)]$. A possible move of *user* is either $<$, $=$ or $>$, but a move must be *consistent* with all constraints induced in the 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) K-rounds at most winning strategy for *prog* in the *GAME(N,M)*.

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! Let us describe this hint in formal terms as an *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 in $[1..(N+M)]$.

Three constraints are absolutely natural: $(1)u+l+h \leq N$, $(2)u+l+h+v = N+M$, $(3)u+l+h \geq 1$. Then we can require that $(4)u \neq 0$ iff $l+h=0$ (since a unique false is among non-tested coins iff all previous balancings gave equal weights), and $(5)v_1 = 0$ or $v_2 = 0$ (since it is not reasonable to add extra valid coins on both pans of a balance). Additional constraints should be imposed on queries (since we can borrow coins for weighing from available non-tested, 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. a 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 $>$ to a query which causes a change in positions

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

in accordance with the query and reply:

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

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

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

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

The final position is a position $(u,\ u,\ h,\ v,\ ((0,0,0,0),(0,0,0,0)))$, where $u+l+h=1$. Thus the game and corresponding abstract model are constructed. An overall amount of positions and moves in $game(N,M)$ is less than $\frac{(N+1)^6}{6}$. And we are ready to present a high-level model-checking-based design for the metaprogram:

1. (a) to input numbers N and M of coins in 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} is valid in the *initial* position, then go to 2, else output impossibility of the strategy and halt;
2. to output a program which model checks 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, namely: for every $i \in [1..K]$ downwards (i.e., from $i=K$ to $i=1$) it outputs to the user a move from the *current position* to an *intermediate position*, where a formula

$\neg fail \wedge [move_B](fail \vee WIN^{i-1}_{win}(false))$ is valid in the abstract model; then it inputs the user's reply $<$, $=$ or $>$ and defines the *next position*.

Correctness of the final high-level design follows from Proposition 2.

4. Testing model checkers via games

Importance of teaching program logics and model checking is due to importance of model checking applications. The main area of model-checking applications is automatic verification of hard- and software presented as finite state systems [11] while automatic model checking verification of high-level software specifications [7, 1] or automatic test generation [18] are rapidly developing new application domains. We suppose that in both cases high-level reliability of model checkers is of extreme importance due to 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 discuss some of the reasons behind this situation. 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 comparison with the size of systems to be verified, while proof checking has a linear complexity. Next, the most popular model checkers SMV [8] and SPIN [20] 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 [11].

Unfortunately, both above reasons are invalid for model checkers of the μ-Calculus in finite models. An approach à la theorem proving is impossible due to exponential complexity of model checking "proofs". At the same time, natural transparency of model checking for temporal logics is lost due to complicated interaction between 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 reasons mentioned above, the polyvariant approach to reliable model checking is time, space and cost expensive. The second approach seems to be problematic since test-generation is a non-trivial problem itself. This problem is addressed in [5] and briefly discussed in the next paragraph. As for formal verification of model checkers, let us point out two recent papers [30, 26]. The first paper [30] has described a model checker generated automatically from a proof. This model checker is a Caml-implementation of a model checking algorithm from [34], it is generated by an interactive logic framework Coq from a formally presented proof of correctness of the algorithm. The second paper [26] has described the formal specification and verification of the efficient algorithm for real-time model checking implemented in the model checker RAVEN. It was specified and verified using the KIV verification system. Thus we can summarize that formal verification of model checkers is a new developing research domain, but not a practical approach to implementing reliable model checkers.

Why 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 (i.e., must have predictable results) and exploit non-trivial combinations of fixpoints. But these two claims are mutually exclusive: predictability of results implies the formulae simplicity, while non-trivial combinations of fixpoints are non-trivial for forecasting. Maybe, the most appropriate solution to overall testing of model checkers is to test them against a formally verified model checker on automatically generated test suits.

As far as manual overall testing of model checkers is concerned, the problem domain of finite games seems to be the best choice for it, since it comprises understandability of formulae and verifiability of

results. Correctness of the results in this case can be checked manually or by means of implementing program robots for player simulation. Below we present two examples of parameterized finite games which were in use for manual testing of model checkers for the μ-Calculus and finite models in the specification and verification project **REAL** [22, 23, 5, 24, 25]. In section 5 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 have used parameterized games for tracking how model checkers react to changes in the model size. All examples in this section are clearly presented as searching problems for a winning strategy in finite games for two players while all examples in section 5 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 "Millennium Game".

> • On the eve of the New Year 19NM (N, M $\in$ [0..9]) Alice and Bob were playing 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.

Another example is a metaprogram problem discussed in section 3. We would like to remark here that all the above examples deal with formulae WIN and $WIN^i_{win}(false)$, $i \geq 0$.

5. More complicated test suits

A class of test suits presented below is more complicated than previous ones since it relies upon a more complicated concept of games with fairness constraints. 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 constraints prohibit sessions where some constraint holds infinitely often: a session meets (satisfies) the constraints C iff every constraint in C holds only finite number of times in the session. In contrast, *fairness conditions* prohibit sessions where some condition holds only finite number of times. 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 meet (satisfy) fairness constraints* is a strategy which guarantees win in every finite session and guarantees that every infinite session is fair to some fairness constraint.

Proposition 3. *Let (P, M_A, M_B, F, C) be a game with fairness constraints, and (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 statements are equivalent:*

- *the strategy is a winning strategy for sessions which meet 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 $\nu q.\big([a]q \wedge \mu r.(p \vee [a]r)\big)$. A sub-formula $\phi \equiv \mu r.(p \vee [a]r)$ of this formula is valid in a model in those states where every infinite a-path eventually leads to p. Another 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 in those states where every a-path always leads to ϕ. Thus the formula $\nu q.\big([a]q \wedge \mu r.(p \vee [a]r)\big) \equiv \nu q.([a]q \wedge \phi)$ is valid in a state of a model iff every infinite a-path infinitely often visits the 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 G be a finite game of two players with fairness constraints C. 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 M_G where every infinite session is fair with respect to $\vee C$;*
- *$FAIRWIN$ is valid in those states of M_G where the player A has a winning strategy in sessions which meet the constraints C.*

Let us present an example of a puzzle 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 square (say, Central Station Square) where he/she hopes to get a generous passenger which is ready to pay as much as the driver asks. Taxi can move from one square to another via a road which connects them. Usually the driver selects roads according to his/her will, but in some squares (these squares are known) occasional passengers order him/her to move along a road according to passenger's choice, which sometimes is a bad, poor road (these roads are known too). But for driver's luck, there is a finite number of occasional passengers which would like to select these bad roads. Problem: Define from what initial squares the driver can reach the desirable square while servicing all orders of all occasional passengers through its rout?

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

References

[1] R. Anderson, P. Beame, W. Chan, D. Notkin, *Experiences with the Application of Symbolic Model Checking to the Analysis of Software Specifications*, Lect. Notes Comput. Sci., **1755**, 1999.

[2] M. Baldamus, K. Schneider, M. Wenz, R. Ziller, *Can American Checkers be Solved by Means of Symbolic Model Checking?*, Proc. of Formal Methods Elsewhere, A Satellite Workshop of FORTE-PSTV-2000, Techn. Rep. 11-00, Comput. Laboratory, Univ. of Kent, Canterbury, UK, 2000, 3–17 (to appear in Electronic Notes in Theor. Comput. Sci.).

[3] J.C. Bradfield, S. Stirling, *Local Model Checking for Infinite State Spaces*, Theor. Comput. Sci., **96**, 1992, 157–174.

[4] S.A. Berezine, N.V. Shilov, *An approach to effective model-checking of real-time finite-state machines in Mu-Calculus*, Lect. Notes Comput. Sci., **813**, 1994, 47–55.

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

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

[7] T. Bultan, R. Gerber, W. Pugh, *Model-Checking Concurrent Systems with Unbounded Integer Variables: Symbolic Representation, Approximation, and Experimental Results*, ACM Trans. on Prog. Lang. and Syst., **21**, No 4, 1999, 747–789.

[8] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, *Symbolic Model Checking: 10^{20} states and beyond*, Information and Computation, **98**, No 2, 1992, 142–170.

[9] O. Burkart, *Automatic Verification of Sequential Infinite-State Processes*, Lect. Notes Comput. Sci., **1354**, 1997.

[10] O. Burcart, B. Steffen *Model-Checking the Full Modal Mu-Calculus for Infinite Sequential Processes*, Lect. Notes Comput. Sci., **1256**, 1997, 419–429.

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

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

[13] R. Cleaveland, M. Klain, B. Steffen *Faster Model-Checking for Mu-Calculus*, Lect. Notes Comput. Sci., **663**, 1993, 410–422.

[14] M. Dam, L. Fredlund, D. Gurov, *Toward Parametric Verification of Distributed Systems*, Lect. Notes Comput. Sci., **1536**, 1998, 150–185.

[15] E.A. Emerson, C.S. Jutla, A.P. Sistla, *On model-checking for fragments of Mu-Calculus*, Lect. Notes Comput. Sci., **697**, 1993, 385–396.

[16] *FM'99 - Formal Methods*, World Congress on Formal Methods in the Development of Computer Systems, Toulouse, France, September 1999, Proc., **I, II**, Wing J.M., Woodcock J., Davies J. (Eds), Lect. Notes Comput. Sci., **1708, 1709**, 1999.

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

[18] A. Gargantini, C. Heitmeyer, *Using Model Checking to Generate Tests from Requirements Specifications*, ACM SIGSOFT Software Engineering Notes, **24**, No 6, 1999, 146–162 (simultaneously Lect. Notes Comput. Sci., **1687**, 1999).

[19] D. Harel *First-Order Dynamic Logic*, Lect. Notes Comput. Sci., **68**, 1979.

[20] G.J. Holzmann, D. Peled, *The state of spin*, Lect. Notes Comput. Sci., **1102**, 1996, 385–389.

[21] D. Kozen, *Results on the Propositional Mu-Calculus*, Theor. Comput. Sci., **27**, No 3, 1983, 333–354.

[22] V.A. Nepomniaschy, N.V. Shilov, *Real92: A combined specification language for systems and properties of real-time communicating processes*, Lect. Notes Comput. Sci., **735**, 1993, 377–393.

[23] V.A. Nepomniaschy, N.V. Shilov, E.V. Bodin *A concurrent systems specification language based on SDL & CTL*, Proc. of Workshop on Concurrency, Specifications & Programming, Humboldt University, Berlin, Informatik-Bericht, No 36, 1994, 15–26.

[24] V.A. Nepomniaschy, N.V. Shilov, E.V. Bodin *A new language Basic-REAL for specification and verification of distributed system models*, A.P. Ershov's Institute of Informatics Systems, Novosibirsk, Report No 65 1999, 39 p. (also available at http://www.iis.nsk.su/edu/publish/bre99/bre99_ps.zip).

[25] V.A. Nepomniaschy, N.V. Shilov, E.V. Bodin *Specification and Verification of Distributed System by means of Elementary-REAL language*, Programming and Computer Software, **25**, No 4, 1999, 222–232.

[26] W. Reif, J. Ruf, G. Schellhorn, T. Vollmer *Do you trust model checker?*, Lect. Notes Comput. Sci., **1954**, 2000, 179–196.

[27] N.V. Shilov, K. Yi, *Engaging Students with Theory through ACM Collegiate Programming Contests*, Accepted for publication in Communications of ACM, 2000, 10 p.

[28] N.V. Shilov, K. Yi, *Puzzles for Learning Model Checking, Model Checking for Programming Puzzles, Puzzles for Testing Model Checkers*, Proc. of Formal Methods Elsewhere, A Satellite Workshop of FORTE-PSTV-2000, Techn. Rep. 11-00, Computer Laboratory, University of Kent, Canterbury, UK, 2000, 18–37 (to appear in Electronic Notes in Theoretical Computer Science).

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

[30] C. Sprenger, *A verified Model Checker for the Modal mu-Calculus in Coq*, Lect. Notes Comput. Sci., **1384**, 167–183.

[31] Steven P., Stirling C. *Practical Model Checking Using Games*, Lect. Notes Comput. Sci., **1384**, 1998, 85–101.

[32] Stirling C. *Local Model Checking Games*, Lect. Notes Comput. Sci., **962**, 1995, 1–11.

[33] Stirling C. *Local Model Checking Games*, Lect. Notes Comput. Sci., **1055**, 1996, 298–312.

[34] Winskel G. *A note on model checking the modal μ-calculus*, Theor. Comput. Sci., **83**, No 2, 1991, 157–167.