A Note on Model Checking Reuse*

N.V. Shilov † 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
{kwang, shilov}@ropas.kaist.ac.kr

Abstract

The propositional μ-Calculus (μC) is a powerful propositional program logic with fixpoints. Some fragments of μC are internal or intermediate logics for a variety of model checkers for state-based propositional temporal/program logics like Computation Tree Logic (CTL) in finite models. But in spite of its expressive power, μC fails to express some very natural properties of finite models (ex., commutativeness of actions), another more expressive logic is required. For it we suggest a Second-Order Elementary Propositional Dynamic Logic (SOEPDL) and demonstrate that this logic is strictly more expressive then μC in finite models. The main problem which we address is how to reuse a model checker for a fragment of μC in finite models for checking SOEPDL formulae in finite models too. We prove a simple sufficient (semantical) criterion for this problem: if an existence of a winning strategy in finite games is expressible in a fragment of μC then a model checker for this fragment can be reused without any change for checking SOEPDL in finite models. The property mentioned above is expressible in μC itself by a simple formula with a unique least fixpoint but is not expressible in CTL. It implies that μC model checkers in finite models can be reused for model checking SOEPDL in finite models too. Nevertheless we do not know whether it is possible to reuse a model checker for CTL in finite models for checking SOEPDL in finite models.

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

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

1 Introduction

The propositional μ-Calculus (μC) is a powerful propositional program logic with fixpoints [11]. Some fragments of μC are internal or intermediate logics for a variety of polynomial model checkers for state-based propositional temporal/program logics like Computation Tree Logic (CTL) [4, 5]. In contrast to CTL, best known model checking algorithms for μC and finite models are exponential. For example, a time bound of Faster Model Checking Algorithm (FMC-algorithm) [6] is roughly $O(m \times f) \times \left(\frac{m \times f}{a}\right)^{a-1}$ where m is a model overall size, f is a formula size and an alternating depth a of a formula is a maximal amount of alternations in fixpoints with respect to the syntactical dependences. A formal definition is out of scope of the paper due to space limitations. We would like only to point out that the alternating depth is not greater then the fixpoint nesting. As far as concerns a complexity class of the model checking problem for μC in finite models then it is known that the problem is in $\mathcal{NP} \bigcap co - \mathcal{NP}$ [7], i.e. it is not more complicated then checking formulae of the propositional calculus to be a tautology and a satisfiable formula. Due to this reason it seems to be very hard to prove an exponential lower bound for the model checking problem for the μC in finite models. Since it is not known whether the problem is complete in $\mathcal{NP}$ then it seems to be more realistic to try to find a polynomial model checking algorithm for μC in finite models. Several expressive fragments of μC which have polynomial hard model checking problem for finite models have been identified [7, 2]. As follows from the upper bound for the FMC-algorithm, formulae with a bounded alternating nesting depth form a fragment of this kind.

But in spite of its expressive power μC fails to express some very natural properties of finite models, ex., commutativeness of actions. Another more expressive logic is required for capturing this property. We suggest for properties of this kind a Second-Order Elementary Propositional Dynamic Logic (SOEPDL) – a variation of μC with second order quantifiers instead of fixpoints. This logic was introduced in [15] but under another name $2M$. A very similar and expressively equivalent logic SOPDL was introduced in [17, 18]: the only difference with SOEPDL is use of nondeterministic program schemata (flowcharts) instead of indivisible action variables. We demonstrate that SOEPDL is strictly more expressive then μC in finite models. In contrast in infinite trees both logics have equal expressive power with Second-Order Logic of monadic Successors [12] since μ is expressible in SOEPDL and it is known [13] that that μC and Second-Order Logic of monadic Successors are expressively equivalent on infinite trees.

The main problem which we address in the paper is how to reuse a model checker for a fragment of μC in finite models for checking SOEPDL formulae in finite models too in spite of lack of expressive power of νC. We suggest a simple sufficient (semantical) criterion for this problem: if an existence of a winning strategy in finite games is expressible in a fragment of μC then a model checker for this fragment in finite models can be reused without any change for checking SOEPDL in finite models. The property mentioned above is expressible in μC itself by a simple formula with a unique least fixpoint but is not expressible in CTL. It implies that μC model checkers in finite models can be reused for model checking SOEPDL in finite models too. Nevertheless we do not know whether it is possible to reuse a model checker for CTL in finite models for checking SOEPDL in finite models.

We would like to remark also that close connections between model checking for μC from one side and special games from another side were extensively examined [14, 15, 16]. In particular, [14] defined *infinite* model checking games and established an equivalence of local model checking to an existence of a winning strategy. Then [15] defined *finite* fixed point games and characterized indistinguishability of processes by means of formulae with bounded amounts of modalities and fixpoints in terms of winning strategies with bounded amounts of moves. Paper [16] exploited model-checking games for practical efficient local model checking processes. [15] defined also a second order propositional modal logic $2M$, corresponding games and established indistinguishability of processes by means of formulae with bounded amounts of modalities and quantifiers in terms of winning strategies with bounded amounts of moves. So it is possible to summarize that [14, 15, 16] exploit infinite games for model checking μC formulae in finite and infinite models. In contrast, we exploit finite games for forcing model checkers for fragments of μC in finite models to check more expressive logic SOEPDL in finite models also.

2 EPDL, μ-Calculus and SOEPDL

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)[9] 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[1]. 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:

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

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

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 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** [3].

Finite games can illustrate all EPDL-related notions. A formal *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 a formula WIN_0 be $false$ and for every $i \geq 1$ let WIN_{i+1} be a formula $\neg fail \wedge \langle move_A\rangle\big(\neg fail \wedge [move_B](fail \vee WIN_i)\big)$. 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.*

- *For every finite game of two players (P, M_A, M_B, F), 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 against B with i-rounds at most.*
- *For every finite game of two players (P, M_A, M_B, F) 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 against B.*
- *An infinite disjunction $\bigvee_{i\geq 0} WIN_i$ is illegal formula of EPDL and is not equivalent to any formula of EPDL.*

Due to space limitation let us sketch a proof for only the last item of the above proposition. For it let us consider a model NEG with all non-positive integers as a domain and interpret $fail$ to be valid only on 0 while interpret $move_{A,B}$ to be a successor function $\lambda x.(x+1)$ on negatives:

$$\ldots \to \overset{-(i+1)}{\circ} \to \overset{-i}{\circ} \to \ \ldots \to \overset{-1}{\circ} \to \underbrace{\overset{0}{\circ}}_{fail}.$$

Then let us define *nesting* for EPDL formulae by induction on formulae structure:

$nest(fail) = nest(true) = nest(false) = 0,$
$nest(\neg\phi) = nest(\phi),$
$nest(\phi \wedge \psi) = nest(\phi \vee \psi) = \max\{nest(\phi), nest(\psi)\},$
$nest([a]\phi) = nest(\langle a\rangle\phi) = 1 + nest(\phi).$

Then for every EPDL formula ϕ, for all $k, l > nest(\phi)$ the following holds: $(-k) \models_{NEG} \phi \Leftrightarrow (-l) \models_{NEG} \phi$. So for every formula of EPDL there exists a non-positive number prior to which the formula is a boolean constant in NEG, while $\bigvee_{i\geq 0} WIN_i$ is valid in all even integers and is invalid in all odd integers.

The above proposition 1 leads to the following suggestion naturally. Let us define *propositional μ-Calculus* (μC) 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[2]. We would like also to impose the following context-sensitive restriction: no bounded negative instance of propositional variables. Informally speaking $\mu p.\phi$ is an "abbreviation" for an infinite disjunction

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

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

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

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

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

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

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

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

Hence the above proposition 1 implies the following

Theorem 1 *Let $\mu\ win.\Big(\neg fail \wedge \langle move_A \rangle \big(\neg fail \wedge [move_B](fail \vee win)\big)\Big)$ be a formula WIN of μC. Then is not equivalent to any formula of EPDL and for every finite game of two players (P, M_A, M_B, F), a formula WIN is valid in those states of $G_{(P,M_A,M_B,F)}$ where a player A has a wining strategy against B.*

The *Second Order Elementary Propositional Dynamic Logic* (SOEPDL) ($2M$ in [15]) is an extension of EPDL with quantifiers over propositional variables and special modalities. The syntax of SOEPDL consists of formulae which are constructed from the above alphabets of propositional and action variables too. Propositional variables are (elementary) formulae. Propositional combinations (negations, conjunctions and disjunctions, etc.) of formulae are formulae. Almost all modalities are associated with action variables like in EPDL or μC. Two special modalities are $\Box$ and $\Diamond$. Quantifiers over propositional variables are admissible too: if p is a propositional variable then $\forall p.$ and $\exists p.$ are quantifier prefixes. The semantics of SOPDL is defined in the same models as for EPDL and μC. Propositional operations and modalities associated with action variables have the usual semantics. For understanding semantics of special modalities let us represent a model as a labeled graph; then $\Box$ and $\Diamond$ are "in all reachable states" and "in some reachable state". The semantics of quantifiers is straightforward from their names – "for every/some interpretation of a propositional variable". Formally speaking, for every model $M = (D_M, I_M)$ the *validity* relation $\models_M$ between states and formulae of EPDL can be extended on formulae of SOEPDL as follows:

5. $s \models_M (\Diamond\phi)$ iff $s' \models_M \phi$ for same state s' reachable from s in M,

6. $s \models_M (\Box\phi)$ iff $s' \models_M \phi$ for every state s' reachable from s in M,

(here s' is reachable from s in M means that there is a finite path from s to s' in M presented as a labeled oriented graph)

7. $s \models_M (\exists p.\phi)$ iff $s \models_{M_{S/p}} \phi$ for some $S \subseteq D_M$,

8. $s \models_M (\forall p.\phi)$ iff $s \models_{M_{S/p}} \phi$ for every $S \subseteq D_M$

(here $M_{S/p}$ is a model which agrees with M everywhere but interpretation of p where $P_{M_{S/p}}(p) = S$).

Theorem 2 *SOEPDL is strictly more expressive then μC:*

- *Every μC formula is equivalent to some SOEPDL formula which can be constructed in a linear time.*
- *SOEPDL formula $\forall p.\big(([a][b]p) \to ([b][a]p)\big)$ is not equivalent in finite models to any μC formula.*

Really, every μC formula can be translated (in a linear time) into an equivalent SOEPDL formula since the following equivalences hold:

- $(\mu p.\phi) \leftrightarrow (\forall p.(\Box(\phi \to p) \to p))$,
- $(\nu p.\phi) \leftrightarrow (\exists p.(\Box(p \to \phi) \wedge p))$.

Then let us compare two models:

$$M_1: \begin{tikzcd} s \arrow[loop left, "a"] \arrow[loop right, "b"] \end{tikzcd} \qquad M_3: \begin{tikzcd} & s \arrow[dl, "a"'] \arrow[dr, "a"] & \\ r \arrow[loop left, "a"] \arrow[loop right, "b"] & & t \arrow[loop left, "a"] \arrow[loop right, "b"] \end{tikzcd}$$

In these models every μC formula is equivalent to a formulae of EPDL, since $|M_{1,3}| \leq 3$. EPDL formulae can not distinguish M_1 and M_3 whether a current state s is in M_1 or in M_2. In contrast, if ϕ is SOEPDL formula $\forall p.\big(([a][b]p) \to ([b][a]p)\big)$ then $s \models_{M_1} \phi$ but $s \not\models_{M_2} \phi$. We would like also to remark that another closely related SOEPDL formula $\forall p.\big(([a][b]p) \leftrightarrow ([b][a]p)\big)$ (which express commutativeness of a and b in a model) is non-expressible in μC too.

3 SOEPDL Model Checking: Win a Game

Model checking [5] 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/Kripke Structures. A *global (model) checking* problem consists in a calculation of *the set of all states* of a model where a formula is valid, while a *local (model) checking* problem 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. 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 μC in 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) [6], 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. In particular, the a complexity of model checking a fixed formula WIN (theorem 1) in a model is linear in a size of the model.

Now we are ready to discuss the main problem which is addressed in the paper, a problem of a model checker reusability:

> You have a model checker for a fragment of μC and finite models.
> How to force it to check SOEPDL formulae in finite models in spite
> of lack of expressive power of μC?

We can not to encode all formulae of SOEPDL in terms of input logic of a model checker due to lack of expressive power of μC. Hence we have to encode SOEPDL formulae in input finite models of this model checker. Basically a hint is construct a special finite game and then check the game for winning strategies.

Let us be concrete. Let ξ be SOEPDL formula. Without loss of generality we can assume that variables bounded by different quantifiers are different (so there are no name collisions). Moreover we can assume that negations in the formula are applied to propositional variables only since the following equivalences hold:
$(\neg(\neg\phi)) \leftrightarrow \phi$,
$(\neg(\phi \wedge \psi)) \leftrightarrow ((\neg\phi) \vee (\neg\psi))$,
$(\neg(\langle a\rangle\phi)) \leftrightarrow ([a](\neg\phi))$,
$(\neg(\Diamond\phi)) \leftrightarrow (\Box(\neg\phi))$,
$(\neg(\exists p.\phi)) \leftrightarrow (\forall p.(\neg\phi))$.
Let x, ... z be a list of all bounded propositional variables in ϕ. A subformula of ξ is a formula, which is (syntactically) a part of ϕ. A subformula is said to be *conjunctive* iff it has one of the following forms: $\phi \wedge \psi$, $[a]\phi$, $\Box\phi$ or $\forall p.\phi$. A subformula is said to be *disjunctive* iff it has one of the following forms: $\phi \vee \psi$, $\langle a\rangle\phi$, $\Diamond\phi$ or $\exists p.\phi$. Propositional variables and their negations (and only

they) are neither conjunctive nor disjunctive subformulae. Let M be a finite model (D_M, I_M). We are going to construct a finite game $G(M,\xi)$ of two player *Spoiler* and *Duplicator*. We would not like to go into technical details of this game presentation. In contrast we would like to describe a draft game $dG(M,\xi)$ only. In this draft game both players have right to make several consequentive moves and they have their own winning states. Let us recall that a formal definition of finite games (in section 2) prohibits several consequentive moves for both players and a win depends of who enters the first a fixed set of final states – a player or a counterpart. We would like to remark also that a draft game $dG(M,\xi)$ can be trivially converted into a formal game $G(M,\xi)$ with linear complexity.

Positions in this game $dG(M,\xi)$ are tuples $(s, S_x, ...S_z, \psi)$ where $s \in D_M$ is a current state, $S_x, ...S_z \subseteq D_M$ are current interpretations for x, ... y, and ψ is a verified subformula of ξ.

A player *Spoiler* has moves of 4 kinds related conjunctive subformulae:

- $(s, S_x, ...S_z, (\psi_1 \wedge \psi_2)) \to (s, S_x, ...S_z, \psi_i)$ for every $i \in \{1,2\}$;
- $(s, S_x, ...S_z, ([a]\psi)) \to (t, S_x, ...S_z, \psi)$ for every t such that $(s,t) \in I(a)$;
- $(s, S_x, ...S_z, (\Box\psi)) \to (t, S_x, ...S_z, \psi)$ for every t reachable from s in M;
- $(s, S_x, ...S_y, ...S_z, (\forall y.\psi)) \to (t, S_x, ...S, ...S_z, \psi)$ for every $S \subseteq D$.

A player *Duplicator* has moves of 4 kinds related disjunctive subformulae:

- $(s, S_x, ...S_z, (\psi_1 \vee \psi_2)) \to (s, S_x, ...S_z, \psi_i)$ for every $i \in \{1,2\}$;
- $(s, S_x, ...S_z, (\langle a\rangle\psi)) \to (t, S_x, ...S_z, \psi)$ for every t such that $(s,t) \in I(a)$;
- $(s, S_x, ...S_z, (\Diamond\psi)) \to (t, S_x, ...S_z, \psi)$ for every t reachable from s in M;
- $(s, S_x, ...S_y, ...S_z, (\exists y.\psi)) \to (t, S_x, ...S, ...S_z, \psi)$ for every $S \subseteq D$.

Duplicator wins in positions of 5 following kinds:

- $(s, S_x, ...S_z, true)$,
- $(s, S_x, ...S_z, p)$ where p is a free propositional variable and $s \in I_M(p)$,
- $(s, S_x, ...S_z, \neg p)$ where p is a free propositional variable and $s \notin I_M(p)$,
- $(s, S_x, ...S_y, ...S_z, y)$, where $s \in I(y)$,
- $(s, S_x, ...S_y, ...S_z, \neg y)$, where $s \notin I(y)$.

Spoiler wins in positions of 5 following kinds:

- $(s, S_x, ...S_z, false)$,

- $(s, S_x, ...S_z, p)$ where p is a free propositional variable and $s \notin I_M(p)$,
- $(s, S_x, ...S_z, \neg p)$ where p is a free propositional variable and $s \in I_M(p)$,
- $(s, S_x, ...S_y, ...S_z, y)$, where $s \notin I(y)$,
- $(s, S_x, ...S_y, ...S_z, \neg y)$, where $s \in I(y)$.

A draft game $dG(M, \xi)$ is over. Intuitively: Duplicator is trying to validate a formula in a current state while a rival Spoiler is trying to falsify the formula in current state. An amount of positions is $O(2^{d\times f} \times d \times f)$. An amount of moves is the same. As we discussed above, $dG(M, \xi)$ can be converted in to a formal finite game $G(M, \xi)$ in a linear time.

Proposition 2 *For every finite model $M = (D_M, I_M)$ and every SOEPDL formula ξ there exists a finite game $G(M, \xi)$ of two players Spoiler and Duplicator with overall size $O(2^{d\times f} \times d \times f)$ such that the following holds for every state $s \in D_M$ and every subformula ϕ of ξ: Duplicator has a winning strategy against Spoiler in a position $(s, I_M(x), ...I_M(z), \phi)$ iff $s \models_M \phi$ (where $x, ...z$ is a list of all bounded variables of ξ). A game can be constructed in time $O(2^{d\times f} \times d \times f)$.*

It immediately implies the following sufficient *reusability criterion*:

Theorem 3 *Let L be a logic with a formula ϕ which expresses an existence of a winning strategy for a player against a counterpart in positions of finite games. Let MC be a model checker for this logic L in finite models which can check this formula ϕ in finite models with time complexity $T(m)$ where m is an overall model size. Let K be another logic which is expressively equivalent to a fragment of SOEPDL and $C(f)$ and $S(f)$ be time and space complexity of a translation K formulae into SOEPDL formulae where f is a formula size. Then MC can be reused for checking K formulae in finite models and upper time bound for model checking K in finite models is $max\{C(f),\ T(O(2^{d\times S(f)} \times d \times S(f)))\}$.*

Let WIN be a formula $\mu\ win.\Big(\neg fail \wedge \langle move_A\rangle(\neg fail \wedge [move_B](fail \vee win))\Big)$ of μC. In accordance with theorem 1, WIN is valid in those positions of a finite game where a player has a winning strategy against a counterpart. In accordance with FMC-algorithm, WIN can be model checked in a time $O(m)$ where m is an overall size of a game (all other parameters are not important since WIN is a fixed formula with $a_{WIN} = 1$ and $F_{WIN} = const$). Combining this upper bound with the above theorem 3 we get the following

Theorem 4 *Let MC be a model checker which implements a FMC-algorithm [6] or another linear in an overall model size model checking algorithm for (at least) a unique formula $\mu\ win.\Big(\neg fail \wedge \langle move_A\rangle\big(\neg fail \wedge [move_B](fail \vee win)\big)\Big)$ of μC in finite models. MC can be reused for checking all formulae of SOEPDL and μC in finite models with upper time bound $O(2^{d\times f} \times d \times f)$ and $O(exp(d \times f) \times d \times f)$ respectively.*

4 Toward reuse of CTL Model Checkers

A basic *branching time* temporal logic called *Computational Tree Logic* (CTL) [4, 5] can be introduced as a fragment of μC with a unique action variable *next* and a very rigorous discipline for fixpoints and a special notation:

- $\mathbf{AX}\phi$ is a CTL notation for $[next]\phi$,
- $\mathbf{EX}\phi$ is a CTL notation for $\langle next\rangle\phi$,
- $\mathbf{AG}\phi$ is a CTL notation for $\nu p.\ (\phi \wedge [next]p)$,
- $\mathbf{AF}\phi$ is a CTL notation for $\mu p.\ (\phi \vee [next]p)$,
- $\mathbf{EG}\phi$ is a CTL notation for $\nu p.\ (\phi \wedge \langle next\rangle p)$,
- $\mathbf{EF}\phi$ is a CTL notation for $\mu p.\ (\phi \vee \langle next\rangle p)$,
- $\mathbf{A}(\phi\mathbf{U}\psi)$ is a CTL notation for $\mu p.\ \big(\psi \vee (\phi \wedge [next]p)\big)$,
- $\mathbf{E}(\phi\mathbf{U}\psi)$ is a CTL notation for $\mu p.\ \big(\psi\ \vee (\phi \wedge \langle next\rangle p)\big)$,

where p is always a new propositional variable. In the CTL framework $\mathbf{A}$ is read as "*for all futures*", $\mathbf{E}$ is read as "*for some futures*", $\mathbf{X}$ is read as "*next time*", $\mathbf{G}$ is read as "*always*", $\mathbf{A}$ is read as "*sometime*"and $\mathbf{A}$ is read as "*until*".

The above definition of CTL in terms of μC describes semantics of CTL on base of μC. Below we would to define semantics of CTL without use of μC since it will give some intuition behind CTL. Let M be a model. A fullpath in a model M is a maximal sequence *seq* of states $s_1...s_is_{i+1}...$ such that $(s_i, s_{i+1}) \in R_M(next)$ for all adjacent states $(s_i, s_{i+1}) \in seq$. If $seq = s_1...s_i...$ is a fullpath then for every finite $i \leq |seq|$ let us denote by seq_i a state s_i and by seq^i a suffix $s_i....$ The *validity* relation $\models_M$ between states and CTL formulae can be defined inductively with respect to the structure of formulae. For boolean constants, propositional variables and propositional connectives the validity relation is defined in a standard way while we have

1. $s \models_M \mathbf{AX}\phi$ iff $seq_2 \models_M \phi$ for every fullpath with $seq_1 = s$,
2. $s \models_M \mathbf{EX}\phi$ iff $seq_2 \models_M \phi$ for some fullpath with $seq_1 = s$,
3. $s \models_M \mathbf{AG}\phi$ iff
   $seq_i \models_M \phi$ for every fullpath with $seq_1 = s$ and every $1 \leq i \leq |seq|$,
4. $s \models_M \mathbf{AF}\phi$ iff
   $seq_i \models_M \phi$ for every fullpath with $seq_1 = s$ and some $1 \leq i \leq |seq|$,
5. $s \models_M \mathbf{EG}\phi$ iff
   $seq_i \models_M \phi$ for some fullpath with $seq_1 = s$ and every $1 \leq i \leq |seq|$,

6. $s \models_M \mathbf{EF}\phi$ iff
$seq_i \models_M \phi$ for some fullpath with $seq_1 = s$ and some $1 \leq i \leq |seq|$,

7. $s \models_M \mathbf{A}(\phi \mathbf{U} \psi)$ iff $seq_i \models_M \phi$ and $seq_j \models_M \psi$
for every fullpath with $seq_1 = s$,
for some $1 \leq j \leq |seq|$ and every $1 \leq i < j$,

8. $s \models_M \mathbf{E}(\phi \mathbf{U} \psi)$ iff $seq_i \models_M \phi$ and $seq_j \models_M \psi$
for some fullpath with $seq_1 = s$,
for some $1 \leq j \leq |seq|$ and every $1 \leq i < j$.

Let us remark also that we can consider CTL formulae without **AG**, **AF**, **EG**, and **EF** due to the following equivalences

$\mathbf{AF}\phi \leftrightarrow \mathbf{A}(true\mathbf{U}\phi)$, $\quad \mathbf{EG}\phi \leftrightarrow \neg\mathbf{AF}(\neg\phi)$,

$\mathbf{EF}\phi \leftrightarrow \mathbf{E}(true\mathbf{U}\phi)$, $\quad \mathbf{AG}\phi \leftrightarrow \neg\mathbf{EF}(\neg\phi)$.

A part of the reason for a success of CTL is based on automatic model checking of specifications expressed on CTL of systems represented as finite models [5]. Branching temporal logic CTL and polynomial model checking algorithms were developed as a new mathematical background for a new verification methodology for finite state systems by E.M. Clarke and E.A. Emerson, J.-P. Queille and J. Sifacis in the early 1980-ies. At fall of 1980-ies model checking researchers encouraged by polynomial complexity of model checking for CTL in finite models, and success of model checking verification experiments for systems of a moderate size had moved on further research topics, such that model checking for more expressive program logics (like μC) in finite huge models with 10^{20} states and beyond. As far as concerns a handling of huge finite models then an advantage of Ordered Binary Decision Diagrams (OBDD) was realized in 1987-92. The most popular modern model checker SMV for CTL and finite models was implemented by combining CTL model checking algorithm with symbolic representation of finite models. The most recent versions of SMV for UNIX, Linux and Windows95 are free available for download [20]. So, the following CTL reusability problem arise naturally due to popularity and efficiency of CTL model checkers:

> How to reuse a model checker for CTL in finite models for more expressive logics (like μC or SOEPDL) in finite models also?

A first move in this case is just to try to implement a sufficient criterion from theorem 3. – Sorry, this approach is unfruitful due to a lack of expressive power of CTL as is demonstrated below.

Proposition 3 *There is no CTL formula which expresses an existence of a winning strategy for a player against a counterpart in positions of finite games.*

Really, let us consider an infinite model NEG (section 2) again:

$$\ldots \to \overset{-(i+1)}{\circ} \to \overset{-i}{\circ} \to \ \ldots \to \overset{-1}{\circ} \to \underbrace{\overset{0}{\circ}}_{fail}.$$

Then let us define *nesting* for CTL formulae without **AG**, **AF**, **EG**, and **EF**:

$nest(fail) = nest(true) = nest(false) = 0,$
$nest(\neg\phi) = nest(\phi),$
$nest(\phi \wedge \psi) = nest(\phi \vee \psi) =$
$= nest(\mathbf{A}(\phi\mathbf{U}\psi) = nest(\mathbf{E}(\phi\mathbf{U}\psi) =$
$= \max\{nest(\phi), nest(\psi)\},$
$nest(\mathbf{AX}\phi) = nest(\mathbf{EX}\phi) = 1 + nest(\phi).$

Then for every CTL formula ϕ, for all $k, l > nest(\phi)$ the following holds: $(-k) \models_{NEG} \phi \Leftrightarrow (-l) \models_{NEG} \phi$. (It is trivially provable by induction on formulae structure.) Hence for every formula of CTL there exists a non-positive number prior to which the formula is a boolean constant in the NEG. while a winning strategy exists in all even integers and does not exists in all odd integers. If nevertheless an existence of a winning strategy in finite games is expressible by a formula of CTL then this formula in NEG should be valid in all even integers and invalid in all odd integers, since for every $n \geq 0$ no CTL formula can distinguish states of a finite model NEG_n

$$\overset{-n}{\circ} \to \overset{(-n+1)}{\circ} \to \ \ldots \to \overset{-1}{\circ} \to \underbrace{\overset{0}{\circ}}_{fail}.$$

from correspondent states of NEG.

Let us summarize: a sufficient reusability criterion fails for CTL; it implies that a reuse of model checkers for CTL and finite models on stronger program logics remains an open problem. In conclusion we would like also to point out that an awareness about an ability to express and check an existence of a winning strategy in finite games is growing in model checking community. For example, paper [1] suggests an ability of checking winning strategies for american checkers on $n \times n$ desk (instead of special examples with a nice OBDD date representation) as a real measure for a power of a model checker. Another paper [19] discusses importance of finite game examples for learning model checking and testing model checkers.

References

[1] Baldamus M, Schneider K., Wenz M., Ziller R. *Can American Checkers be Solved by Means of Symbolic Model Checking?* in Proc. of Formal Methods Elsewhere. A Satellite Workshop of FORTE-PSTV-2000. Technical Report 11-00, Computer Laboratory, University of Kent at Canterbury, UK, 2000, p.3-17 (to appear in Electronic Notes in Theoretical Computer Science).

[2] 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.

[3] 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.

[4] 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.

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

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

[7] 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.

[8] Fisher M.J. Ladner R.E. *Propositional dynamic logic of regular programs.* J. Comput. System Sci., v.18, n.2, 1979, p.194- 211.

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

[10] Harel D. *Dynamic Logic.* Handbook of Philosophical Logic, v.II, (1-st ed.), Kluwer Academic Publishers, 1994 (2-nd ed.) p.498-604.

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

[12] Rabin M.O. *Decidability of second order theories and automata on infinite trees.* Trans. Amer. Math. Soc., v.141, 1969, p.1-35.

[13] Schlingloff H. *On expressive power of Modal Logic on Trees.* LNCS, v.620, 1992, p.441-450.

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

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

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

[17] Shilov N.V. *Propositional Dynamic Logic with Fixed Points: algorithmical tools for verification of finite state machines.* LNCS, v.620, 1992, p.452-458

[18] Shilov N.V. *Program schemata vs. automata for decidability of program logics.* Theoretical Computer Science, v.175, n.1, 1997, p.15-27

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

[20] http:/www-cad.eecs.berkley.edu/ kenmcmil/smv/