Program Schemata Technique to Solve Propositional Program Logics Revised*

Nikolay Shilov

A.P. Ershov Institute of Informatics Systems, Russian Academy of Sciences
Lavren'ev av. 6, 630090 Novosibirsk, Russia
shilov@iis.nsk.su

Abstract. Propositional program (dynamic, temporal and process) logics are basis for logical specification of program systems (including parallel, distributed and multiagent systems). Therefore development of efficient algorithms (decision procedures) for validation, provability and model checking of program logics is an important research topic for the theory of programming.
The essence of a program schemata technique consists in the following. Formulas of a program logic to be translated into uninterpreted nondeterministic monadic flowcharts (so called Yanov schemata) so that the scheme is total (i.e. terminates) in all special interpretations if and only if the initial formula is a tautology (i.e. is identically true). Since this generalized halting problem is solvable (with an exponential complexity), it implies the decidability of initial program logic (and leads to a decidability upper bound).
The first version of the technique was developed by Nikolay V. Shilov and Valery A. Nepomnjaschy in 1983-1987 for variants of Propositional Dynamic Logic (PDL). In 1997 the technique was expanded on the propositional μ-Calculus. In both cases a special algorithm was used to solve the generalized halting problem.
A recent development of program schemata technique consists in revised decision procedure for the halting problem. A new decision procedure consists in model checking of a special fairness property (presented by some fixed μ-Calculus formula) in finite models presented by Yanov schemata flowcharts. Exponential lower bound for transformation of μ-Calculus formulas to equivalent guarded form is a consequence of the new version of the decision procedure.

1 Propositional μ-Calculus

Let us define syntax and semantics of the propositional μ-Calculus (μC) [8], one of the most expressive propositional program logics.

Definition 1. *Let $Con = \{true, false\}$ be Boolean constants and Var and Act be disjoint (countable) alphabets of propositional and program variables. Syntax of μC consists of formulas to be defined by structural induction.*

* This work is supported by the RFBR-grant # 13-01-00645-a.

– *All constants in Con and variable in Var are formulas.*
– *Any propositional combination of formulas is a formula:* negation $\neg\phi$, conjunction $(\phi \wedge \psi)$ *and* disjunction $(\phi \vee \psi)$ *are formulas for any formulas* ϕ *and* ψ.
– *For any program variable* $a \in Act$ *and formula* ϕ *modal constructs* sometimes $(\langle a\rangle\phi)$ *and* always $([a]\phi)$ *are formulas.*
– *For any propositional variable* $x \in Var$ *and formula* ϕ *without negative*[1] *and bound*[2] *instances of* x*, the* least *fix-point* $(\mu x.\phi)$ *and the* greatest *fix-point* $(\nu x.\phi)$ *are formulas too*[3].

Let us drop the top-level parenthesis around formulas as well as those parenthesis inside formulas that may be restored according to the standard precedence rules for the propositional connectives: negation precedes conjunction that precedes disjunction

For any syntactic expression r and any two expressions of same kind[4] s and t let $r_{t/s}$ be the result of instantiation of t instead of all instances of s in r; also let $r^0_{t/s}$ denote the expression t itself, and for every $n \geq 0$ let $r^{n+1}_{t/s}$ be $r_{r^n_{t/s}/s}$ i.e. $r^m_{t/s}$ stays for m-times substitution of t instead of s in ϕ. For example, if ϕ is a formula $\psi \vee \langle a\rangle x$ (where $x \in Var$ and $a \in Act$ are propositional and program variables) then[5]

– $\phi^0_{true/x} \equiv true$,
– $\phi^1_{true/x} \equiv (\psi \vee (\langle a\rangle true))$,
– $\phi^2_{true/x} \equiv (\psi \vee (\langle a\rangle(\psi \vee (\langle a\rangle true))))$, etc.

μC semantics is defined in models that are called *labeled transition systems* (LTS) in Computer Science and *Kripke systems/structures* in Logic and Philosophy; let us just use term *model* instead of both in this paper.

Definition 2. *Each model* M *is a triple* (D, R, E) *where*

– *the* domain $D \neq \varnothing$ *is a set of* states*;*
– *the* interpretation $R : Act \to 2^{D\times D}$ *is a function that assigns a binary relation* $R(a) \subseteq D \times D$ *to each* $a \in Act$*;*
– *the* valuation $E : Var \to 2^D$ *is a function that assigns a unary predicate* $E(x) \subseteq D$ *to each* $x \in Var$.

If $M = (D, R, E)$ *is a model,* $S \subseteq D$ *is a set of states, and* $x \in VAR$ *is a propositional variable then let* $M_{S/x}$ *denote a model* $(D, R, E_{S/x})$ *where*[6] *valuation*

[1] An instance of a subformula is said to be negative if it is in the scope of odd number of negations; otherwise the instance is said to be positive.
[2] An instance of a variable x is said to be bound if it is in the scope of μx or νx; otherwise it is said to be free.
[3] The definition implies that all bound variable within a formula must be different.
[4] i.e. both are simultaneously propositional variables, program variables, formulas, etc.
[5] Hereafter we use '$\equiv$' for syntax identity, but '$=$' for (set-theoretic) equality.
[6] Acronym *upd* stays for *update*, i.e. the following second-order function modifier: for any function $f : X \times Y$, elements $x \in X$ and $y \in Y$ let $upd(f, x, y) = \lambda z \in X.\ if\ z = x\ then\ y\ else\ f(y)$.

$E_{S/x}$ is $upd(E, x, S)$ i.e. a valuation that may differ from E for propositional variable x only: $E_{S/x}(x) = S$.

Definition 3. *μC semantics in may be defined by extending valuations (provided by models) from propositional variables onto all formulas by structural induction as follows. For any model $M = (D, R, E)$ let*

- *$M(true) = D$, $M(false) = \varnothing$, and $M(x) = E(x)$ for every $x \in Var$;*
- *$M(\neg\phi) = D \setminus M(\phi)$, $M(\phi \wedge \psi) = M(\phi) \cap M(\psi)$, $M(\phi \vee \psi) = M(\phi) \cup M(\psi)$;*
- *$M(\langle a \rangle \phi) = \{s \in D \ : \ \exists t \in D((s,t) \in R(a) \text{ and } t \in M(\phi))\}$,*
- *$M([a]\phi) = \{s \in D \ : \ \forall t \in D((s,t) \in R(a) \text{ implies } t \in M(\phi))\}$;*
- *$M(\mu x.\phi)$ is the least (w.r.t. $\subseteq$) set of states $S \subseteq D$ that $M_{S/x}(\phi) = S$;*
- *$M(\mu x.\phi)$ is the greatest (w.r.t. $\subseteq$) set of states $S \subseteq D$ that $M_{S/x}(\phi) = S$.*

Satisfiability $\models$ *is a ternary relation between states, models and formulas defined as follows: $s \models_M \phi$ iff $s \in M(\phi)$.*

The above definition needs some justification since it refers to existence of the set-theoretic least and greatest fix-points $M_{S/x}(\phi) = S$ over 2^D. Correctness of this definition follows from monotonicity of a function $\lambda S \subseteq D.M_{S/x}(\phi) : 2^D \to 2^D$ (assuming ϕ hasn'tt negative instances of x) and Knaster-Tarski fix-point theorem [13, 8, 15]. We skip full details of this justification (due to space limitations) but formulate a corollary that follows from the proof of the theorem[7].

Corollary 1. *For any propositional variable $x \in Var$, μC-formula ϕ without bound and negative instances of x, for every $n \geq 0$ and every model M the following inclusions hold: $M(\phi^n_{false/x}) \subseteq M(\mu x.\phi)$ and $M(\nu x.\phi) \subseteq M(\phi^n_{true/x})$.*

Definition 4. *Let ϕ be any μC-formula.*

- *ϕ is said to be valid in a model M ($\models_M \phi$), if $M(\phi) = D_M$ where D_M is the domain of the model; ϕ is said to be valid ($\models \phi$), if it is valid in all models.*
- *ϕ is said to be satisfiable in a model M, if there exists a state s such that $s \models_M \phi$; ϕ is said to be satisfiable if it is so in some model.*

The following definition just recalls some general logic concepts.

Definition 5.

- Calculus *is a formal language provided with syntax-driven inference system; if the inference system has axioms (i.e. premise-free inference rules) then provable sentences are those of the language that may be inferred from axioms. A calculus with axioms is called* axiomatic system.
- *A formal language provided with model-base concept of* validity *(for its sentences) is*
  - decidable *if there exists an algorithm to solve the set of valid sentences;*
  - axiomatizeable *if there exists an algorithm to enumerate all valid sentences.*

[7] We need the corollary for justification of some statements in the paper.

– *A calculus (or axiomatic system) provided with model-base validity is*
  • sound *if all provable sentences are valid;*
  • complete *if all valid sentences are provable.*
– *A formal language provided with model-base validity is said to be* syntactically-axiomatizable *if it has a sound and complete axiomatic system.*

In the original paper [8] D. Kozen defined syntax, Kripke semantics (i.e. model-based validity) and axiomatic system (i.e. a calculus itself) for μ-Calculus and proved soundness of the axiomatic system. The first sound and complete axiomatization for μC was built 10 years later by I. Walukiewicz [14]; I. Walukiewicz proved completeness of the original axiomatization next 7 years later in [15].

μ-Calculus was proved to be decidable with exponential upper bound 15 years after the original paper [8] independently by N.V. Shilov in [11] and by E.A. Emerson and C.J. Jutla [3]. E.A. Emerson and C.J. Jutla proved exponential upper bound by reduction of the satisfiability problem to the emptiness problem for Büchi automata; furthermore they also proved $EXP - Time$ completeness of the satisfiability problem. In contrast, N.V. Shilov suggested linear time translation of μC-formulas to non-deterministic Yanov[8] schemata [9, 11] such that reduces validity problem to a so-called generalized halting problem for schemata; it had been shown earlier by V.A. Nepomniaschij and N.V. Shilov [9, 11] that the generalized halting problem is decidable for Yanov schemata in exponential time.

2 Special classes of models and formulas

Definition 6. *μC-formulas ϕ and ψ are said to be equivalent in a class of models $\mathbb{M}$ if $M(\phi) = M(\psi)$ for every model M in this class. In particular, when $\mathbb{M}$ is the class of all models then the formulas are said equivalent. If a class $\mathbb{M}$ consists of a single model M then formulas are said to be equivalent in this model M.*

Definition 7. *μC-formula is said to be* normal *if all instances of negation are at literal level (i.e. $\neg$ may appear only in front of propositional variables in the formula).*

According to the following very standard statement each μC-formula may be transformed in linear time into an equivalent normal formula.

Proposition 1. *For any propositional variable x and program variable a, for any μC-formulas ϕ and ψ the equivalences in the table 1 are valid.*

Definition 8. *An instance of a propositional variable in a formula is said to be guarded if it occurs in the range of any modality $[\ldots]$ or $\langle\ldots\rangle$. A propositional variable is said to be guarded in a formula if all its instances are guarded in the formula. μC-formula is said to be guarded if all its bound variables are guarded.*

[8] Alternative spelling: Ianov.

$\neg(\neg\phi)$ is equivalent to ϕ	
$\neg(\phi \wedge \psi)$ is equivalent to $(\neg\phi) \vee (\neg\psi)$	$\neg(\phi \vee \psi)$ is equivalent to $(\neg\phi) \wedge (\neg\psi)$
$\neg(\langle a\rangle\psi)$ is equivalent to $[a](\neg\psi)$	$\neg([a]\psi)$ is equivalent to $\langle a\rangle(\neg\psi)$
$\neg(\mu x.\psi)$ is equivalent to $\nu x.(\neg\psi_{\neg x/x})$	$\neg(\nu x.\psi)$ is equivalent to $\mu x.(\neg\psi_{\neg x/x})$

Table 1. Normalization equivalences

One can read in papers [14, 15] that every μC-formula can be converted into equivalent guarded one in *polynomial* time. But recently it was proved that *"known guarded transformations can cause an exponential blowup in formula size, contrary to existing claims of polynomial behavior"* [1].

Proposition 2. *Every μC-formula is equivalent to some guarded μC-formula that may be constructed from the input formula in exponential time and space.*

Proof. First, if a formula ϕ has no instances of a propositional variable x then $\mu x.\phi$ and $\nu x.\phi$ are both equivalent to ϕ.

Next, let x be a propositional variable and ϕ be a formula without negative instances of x; let us classify instances of x in ϕ as follows:

- guarded instances or instances in the scope of any *alien* fix-point construct[9];
- unguarded instances that are out of scope of any alien fix-point construct;

let us refer instances of the first type by x_{fp} and instances of the second type by x_{os}; then following equivalences hold:

- $\mu x.\phi$ is equivalent to $\mu x.\phi_{false/x_{os}}$, i.e. the same formula where all unguarded instances of x are replaced by $false$;
- $\nu x.\phi$ is equivalent to $\nu x.\phi_{true/x_{os}}$, i.e. the same formula where all unguarded instances of x are replaced by $true$.

Using these equivalences one can eliminate all unguarded bound instances of variables that are out of scope of any alien fix-point construct.

Finally, for any propositional variables x and y, any μC-formulas $\phi(x, y)$ and $\psi(x, y)$ without negative and bound instances of x and y, for any fix-point constructs $\pi, \rho \in \{\mu, \nu\}$, any fresh[10] propositional variable z and t, the following formulas are equivalent:

- $\pi x.\phi(x, \rho y.\psi(x, y))$,
- $\pi x.\phi(x, \rho y.\psi(\pi z.\phi(z, \rho t.\psi(z, t)), y))$,
- $\pi x.\phi(x, \rho y.\psi(\pi z.\phi(z, y), y))$.

Observe, that if formula $\psi(x, y)$ has no unguarded instances of y, then the formula $\pi x.\phi(x, \rho y.\psi(\pi z.\psi(z, y), y))$ has no unguarded instances of neither x, nor y, nor z in the scope of an alien fix-point construct ρy or πz. ■

[9] i.e. a construct that bounds another variable

[10] i.e. that are not in use neither in ϕ nor in ψ

Definition 9. *A model $M = (D, R, E)$ is said to be strict if every program variable $a \in Act$ is interpreted as a total function $R(a) : D \to D$. A variant of μ-Calculus (with same syntax as μC) based on (i.e. that uses) strict interpretations only, is called strict μ-Calculus (μ-Strict or μS).*

Let us remark that propositions 1 and 2 are true for μ-Strict. But strict μ-Calculus also has some specifics: for every program variable $a \in Act$ and any μS-formula ϕ the following μS-formulas $[a]\phi$ and $\langle a\rangle\phi$ are equivalent due to interpretation of program variables by total functions. This observation implies the following corollary.

Corollary 2. *Every μS-formula is equivalent to some normal guarded box-free*[11] *μS-formula that may be constructed from the input formula in exponential time and space.*

Definition 10. *Formulas (in different languages maybe) are said to be* equally valid *if they all are simultaneously valid (according to their semantics) or all simultaneously are not valid. Similarly, formulas are said to be* equally satisfieble *if they all are simultaneously satisfieble (according to their semantics in different models maybe) or all simultaneously are not satisfieble.*

Let us introduce (classical propositional) implication $\to$ and equivalence $\leftrightarrow$ in the standard manner as macros legal to use in *non-normal* formulas: for any formulas ϕ and ψ let $\phi \to \psi$ stays for $(\neg\phi) \vee \psi$, and $\phi \leftrightarrow \psi$ stays for $(\phi \to \psi) \wedge (\psi \to \phi)$.

Next let us introduce few more macros inspirited by Propositional Dynamic Logic (PDL)[5]: for any program variables a and b, any formulas ϕ and ψ let

- $[(a;\phi?)^*;b]\psi$ stays for $\nu x.([b]\psi \wedge [a](\phi \to x))$,
- $\langle(a;\phi?)^*;b\rangle\psi$ stays for $\mu x.(\langle b\rangle\psi \vee \langle a\rangle(\phi \wedge x))$.

It is possible to say that ';' is *sequential composition* of programs, '*' is non-deterministic *iteration* of a program, and '?' is a *test* construct that converts a 'property' to a guard. Thus formula $[(a;\phi?)^*;b]\psi$ suggests to iterate a any (non-deterministic) number of times (while ϕ holds after each iteration), then apply b and check that at the end ψ is always true; in contrast, formula $\langle(a;\phi?)^*;b\rangle\psi$ suggests to iterate a some (non-deterministic) number of times (with care about ϕ after each iteration), then apply b and eventually ψ should be true.

The following proposition has been proved in [11].

Proposition 3. *Let ϕ be a μC-formula. For every program variable $a \in Act$ that occurs in ϕ, let f_a, g_a and p_a be fresh (disjoint) program variables and propositional variable (individual for each a). Let μS-formula ψ be result of replacement in ϕ of all instances of each program variable $a \in Act$ by an instance of expression $(f_a; p_a?)^*; g_a$; then μC-formula ϕ is equally valid/satisfieble with μS-formula ψ. It implies that every μC-formula is equally valid/satisfieble with some μS-formula that may be constructed from the input formula in linear time.*

[11] i.e. a formula without instances of modality [...]

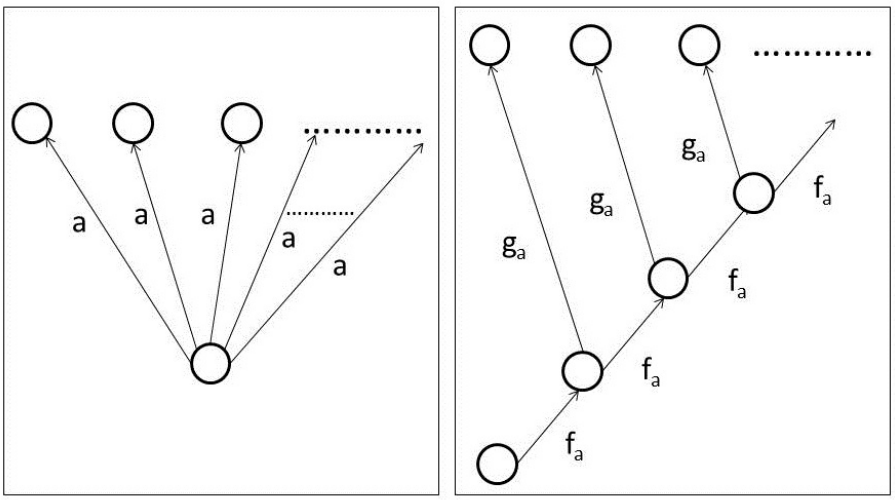

Fig. 1. Simulation of a countable μC-model by μS-model

Proof. Firstly let us remark that ϕ is valid iff it is valid in all *tree-based* countable models. Here a tree-based countable model is any model (D, R, E) where D as the set of nodes and $\{R(a) : a \in Act$ occurs in $\phi\}$ as the set of edges form a directed tree. One can think about a tree-based model for ϕ as a labeled directed tree where nodes are states, edges are graphs of interpreted program variables that occur in ϕ; edges of this tree are marked by corresponding program variables and nodes are marked by propositional variables occurring in ϕ that are evaluated as valid in these nodes. A tree-based model that "emulates" a given model for ϕ can be constructed as follows: for each state glue together all its successor states that are indistinguishable by any formula of μC, and then unfold this reduced model into an infinite tree (starting from any desired state). Let us remark that gluing indistinguishable states and coping by unfolding don't change validity of any formula in states and their copies. Remark also that after gluing together all indistinguishable states the reduced model is (not more than) countable; hence unfolded tree-based model is also countable.

Next, we can transform a countable tree-based model for ϕ into a countable tree-based μS-model for μS-formula ψ, by "simulating" interpretation of each program variable $a \in Act$ (that occurs in ϕ) as specified below and illustrated on Fig. 1: for each state s

- let us introduce a countable set of auxiliary states,
- let f_a enumerates all the auxiliary states in some order,
- let p_a be valid as many times on auxiliary states as is the number of a-successors of s,
- let g_a returns a corresponding a-successor of s for each of auxiliary states.

In the resulting μS-model the validity of the formula ψ in the states of the original μC-model coincides with the validity of ϕ in the original μC-model. ■

The following statement is a corollary from Knaster-Tarski fix-point theorem.

Corollary 3. *For any propositional variable $x \in Var$, any μS-formula ϕ that has no negative or bound instances of x, for any strict model M the following holds:*

- $M(\mu x.\phi) = \bigcup_{n\geq 0} M(\phi^n_{false/x})$,
- $M(\nu x.\phi) = \bigcap_{n\geq 0} M(\phi^n_{true/x})$.

Proof. Both equalities are very similar, so let us prove the first one only by simultaneous induction by number of fix-point constructs in ϕ; since proof of the induction basis and the induction step are very similar, let us just sketch the induction basis, i.e. to prove that $M(\mu x.\phi) = \bigcup_{n\geq 0} M(\phi^n_{false/x})$ for a fix-point-free formula ϕ.

Firstly, subsumption $\bigcup_{n\geq 0} M(\phi^n_{false/x}) \subseteq M(\mu x.\phi)$ is valid in each model according to Corollary 1. Next, let us assume (in contrary) $\bigcup_{n\geq 0} M(\phi^n_{false/x}) \neq M(\mu x.\phi)$. Then let us denote $\bigcup_{n\geq 0} M(\phi^n_{false/x})$ by S. Due to the assumption, $M_{S/x}(\phi) \neq S$. Hence there exists a state $s \in D_M$ such that $s \in (M_{S/x}(\phi) \setminus S)$. Since ϕ is fix-point-free and M is strict, then there exists a finite set $S_{fin} \subseteq S$ such that $s \in M_{S_{fin}/x}(\phi)$. It implies that $s \in \bigcup_{n\geq 0} M(\phi^n_{false/x})$ due to finiteness of S_{fin}. — Contradiction. Hence $M_{S/x}(\phi) = S$. ■

Definition 11. *A model (D, R, E) is called* Herbrand model *[9] (or* free model *in Russian tradition [4, 7]) if the domain D is the set Act^* of all finite words constructed from program variables (including the empty word θ) and interpretation R is defined as follows: $R(a)(w) = wa$ for any program variable $a \in Act$ and any word $w \in Act^*$.*

If to analyze the proof of proposition 3 above then it is possible to prove the following statement.

Proposition 4. *μS-formula is valid/satisfiable iff it is valid/satisfiable in every/some Herbrand model.*

3 Non-deterministic Yanov Schemata

Yanov schemata [6, 4, 10] is one of classical program models that enjoys decidability of many algorithmic problems like (functional) equivalence, emptiness and totality (halting) [4, 7]. The term *Yanov scheme* was introduced by Andrey P. Ershov; he also developed graphical flowchart notation and complete graphical axiomatization for the equivalence problem [4, 10].

Non-deterministic Yanove schemata were introduced in [9] and then were used in [11]. Let us repeat below basic syntax and semantics definitions for non-deterministic Yanov schemata.

Definition 12. *Let us use natural numbers (including 0) as labels*[12]*. Assignment (or labeled assignment operator) is any expression of the form "*$l:\ f$ *goto* L*" where* l *is a label,* f *is a program variable, and* L *is a finite set (the empty set maybe) of labels*[13]*. Choice (or labeled choice operator) is any expression of the form "*$l:\ if\ p\ then\ L^+\ else\ L^-$*" where* l *is a label,* p *is a propositional variable,* L^+ *and* L^- *are finite sets of labels (each may be empty). Non-deterministic Yanov scheme is a finite set of labeled operators.*

According to the definition above, a label may mark several operators in a non-deterministic Yanov scheme. Definition of syntax of the classical Yanov schemata results from the above definition by imposing the following additional constraints:

- a label may mark a single operator in a scheme;
- all sets L, L^+ and L^- in operators are singletons.

Let us reserve term *Yanov scheme* (or simply *scheme*) for non-deterministic Yanov scheme, but use term *standard* Yanov scheme when we discuss the classical case.

Semantics of Yanov schemata is defined in strict models. Speaking informally, *run* of a scheme in a model may start from any state but the first operator to fire must be marked by label 0 (zero); then run consists of firings of operators according to control flow that is defined by labels; run halts when control is passed to any label that does not mark any operator within the scheme[14].

Definition 13. *Let* S *be an arbitrary fixed Yanov scheme and* $M = (D, R, E)$ *be any fixed strict model. Configuration is any pair of the form* (l, s) *where* l *is a label (that occurs in* S*) and* s *is a state (of the model). Firing of an operator is a pair of configurations* $(l, s)(l', s')$ *as defined below:*

- *firing of an assignment operator "*$l:\ f$ *goto* L*" (that occurs in* S*) is a pair of configurations* $(l, s)(l', s')$ *where* $s' = R(f)(s)$ *and* $l' \in L$*;*
- *firing of a choice operator "*$l:\ if\ p\ then\ L^+\ else\ L^-$*" (that occurs in* S*) is a pair of configurations* $(l, s)(l', s')$ *where* $s' = s$ *and* $l' \in \begin{cases} L^+ & \text{if } s \in E(p) \\ L^- & \text{otherwise.} \end{cases}$

A run *is any sequence of configurations* $(l_0, s_0) \dots (l_k, s_k)(l_{k+1}, s_{k+1}) \dots$ *such that each neighbor pair* $(l_k, s_k)(l_{k+1}, s_{k+1})$ *within this sequence is a firing of some operator (of* S *of course). A complete run starts with a configuration with label 0 and is either infinite or ends with a configuration with a final label (in* S*).*

Let us recall some facts and concept about the standard Yanov schemata. Every standard Yanov scheme in any strict model for any initial state has a

[12] Let us assume that notation for number representation is fixed.

[13] Let us use the standard representation for finite sets: $\varnothing$ for the empty set and elements enumerated in a pair of curly parenthesis '{' and '}'.

[14] These labels are called final labels of the scheme.

single complete run that starts in this state. A standard Yanov scheme is said to be *total*, if it hasn't any infinite complete run in any strict model. The *halting (totality) problem* for standard Yanov schemata is to decide for an input scheme whether it is total or not. It is well-known that the problem is decidable, and a scheme is total iff in every Herbrand model starting in configuration[15] $(0, \theta)$ the scheme always halts[16] [7].

Definition 14. *Let $Fin \subseteq Var$ be a finite set of propositional variables. Let us say that a Herbrand model $M = (D, R, E)$ fits Fin, if for every propositional variable $p \in Fin$ its evaluation $E(p)$ is a finite set. A scheme S is said to be total with respect to Fin, if in every Herbrand model that fits Fin, scheme S has a* finite complete *run starting in configuration $(0, \theta)$.*

Definition 15. *Generalized halting (totality) problem for non-deterministic Yanov schemata is to decide for an input scheme S and input finite set of propositional variables Fin whether S is total with respect to Fin.*

Generalized halting problem has been proven to be decidable [9, 11] with upper bound $\exp(n_A + n_C)$ where n_A and n_C are numbers of assignments and choices in the input scheme. Below we will present a new decision procedure for the problem in a sub-class of *guarded schemata* inspirited by a decision procedure for a special class of automata [12].

Definition 16. *Yanov scheme is said to be guarded if any non-empty path on flow-chart of the scheme that starts in a choice operator with some propositional variable as the condition, and ends in a choice operator*[17] *with the same condition, has an instance of an assignment operator.*

It is well-known that functional equivalence is decidable for the standard Yanov schemata [4, 7]. This equivalence may be expanded onto non-deterministic Yanove schemata by considering input-output relations augmented by looping, and can been proven to be decidable since every scheme can be effectively transformed to appropriate equivalent canonical guarded scheme [9]. These equivalence and transformation were very helpful for proving decidability of Propositional Dynamic Logic [9] but aren't so helpful in study of decidability of μ-Calculus.

Nevertheless we are interested in guarded non-deterministic schemata since such schemata may be converted to models for μ-Calculus.

Definition 17. *Let S be a guarded scheme. For each propositional variable p that occurs in S, let $+p$ and $-p$ be a pair of new (fresh) program variables. Let M_S be the following model (D_S, R_S, E_S).*

– *D_S is union of the following three sets:*

[15] Recall that θ is the empty word.
[16] i.e. it has a *finite complete* run
[17] maybe the same operator where the path starts

- $\{(l, f)$: *l is a label, f is a program variable such that S has an assignment "l : f goto ..."* $\}$;
- $\{(l, +p), (l, -p)$: *l is a label, p is a propositional variable such that S has a choice "l : if p then ... else ..."*$\}$;
- $\{(l, stop)$: *l is a final label in S*$\}$.

– *R_S interprets old and new program variables as follows:*
- *for each program variable f that occurs in S, $R_S(f) = \{((l, f), (k, m))$: $(l, f), (k, m) \in D_S$ and S has an assignment "l : f goto L", where $k \in L\}$;*
- *for each propositional variable p in S, $R_S(+p) = \{((l, p), (k, m))$: $(l, f), (k, m) \in D_S$ and S has a choice "l : if p then L^+ else ...", where $k \in L^+\}$;*
- *for each propositional variable p in S, $R_S(-p) = \{((l, p), (k, m))$: $(l, f), (k, m) \in D_S$ and S has a choice "l : if p then ... else L^-", where $k \in L^-\}$.*

– *For each propositional variable p in S, $E_S(p) = \{(l, +p) : (l, +p) \in D_S\}$.*

Any state of D_S in the form $(0, \ldots)$ is called initial.

Definition 18. *Let $Fun \subseteq Act$ be a finite set of program variables, ϕ be μC-formula, $M = (D, R, E)$ be a model, and $s \in D$ be a state. An infinite sequence of states $s_0, s_1, \ldots \subseteq D$ is generated by Fun from s in M, if $s_0 = s$ and for all $k \geq 0$ there is a program variable $a \in Fun$ that $(s_k, s_{k+1}) \in R(a)$. Let us say that*

– *formula ϕ is* inevitable *for Fun in M from s, if every infinite sequence of states $s_0, s_1, \ldots \subseteq D$ generated by Fun from s in M has a state $s_j \models_M \phi$;*
– *program variables Fun are* fair *for (or with respect to) ϕ in M from s, if every infinite sequence of states $s_0, s_1, \ldots \subseteq D$ generated by Fun from s in M has an infinite subsequence $t_0, t_1, \ldots$ that $t_j \models_M \phi$ for all $j \geq 0$.*

Proposition 5. *Let S be a guarded scheme, Fin be a set of propositional variables, M_S be model constructed from S as specified in the definition 17, and Fun be the set of all program variables that correspond to propositional variables in Fin according to this definition. Then the following clauses are equivalent:*

– *S is total with respect to Fin;*
– *Fun is fair with respect to $\bigvee_{p \in Fin} p$ in M_S from every initial state.*

Proof. A path $l_0, \ldots l_k, \ldots$ in a flowchart is said to be valid if there exists a strict model M and sequence of states $s_0, \ldots s_k, \ldots$ (with same length as the path) such that $(l_0, s_0) \ldots (l_k, s_k) \ldots$ is a run in this model. A scheme is said to be *free* [7, 11] if any path on its flowchart is a valid. It is easy to see that every guarded scheme is free. Due to this freedom of S and finiteness of Fin we have: S isn't total with respect to $Fin \Leftrightarrow$ there exists an infinite path in M_S where formula $\bigvee_{p \in Fin} p$ is valid an infinitely often. ■

Definition 19. *Let $Fun \subseteq Act$ be a finite set of program variables, and ϕ be any μC-formula. Let us introduce two more macros:*

– *let*[18] *$AF(Fun, \phi)$ stays for $\mu y.(\phi \vee \bigwedge_{a \in Fun}[a]y)$,*

[18] *AF* means *Always in Future* is a modality from Computation Tree Logic CTL [2].

– *and* $fair(Fun, \phi)$ *stays for* $\nu x.AF(Fun, \phi \wedge x)$.

Proposition 6. *For every model* $M = (D, R, E)$*, for every state* $s \in D$*, every finite set of program variables* Fun*, and every* μC*-formula* ϕ *the following equivalences hold:*

– $s \models_M AF(Fun, \phi) \Leftrightarrow \phi$ *is inevitable for* Fun *in* M *from* s*;*
– $s \models_M fair(Fun, \phi) \Leftrightarrow Fun$ *is fair for* ϕ *in* M *from* s.

Proof. The first equivalence is trivial and well-know from Computational Tree Logic [2]. To prove the second equivalence, let us pickup an arbitrary infinite sequence of states $s_0, s_1, \ldots \subseteq D$ generated by Fun from s in M such that $s_0 \models_M fair(Fun, \phi)$; according to the first equivalence, the sequence has a state $t_0 \models_M \phi \wedge AF(Fun, fair(Fun, \phi))$, i.e. $t_0 \models_M \phi$ and $t_0 \models_M AF(Fun, fair(Fun, \phi))$; it implies that the sequence has another state t_1 (somewhere after t_0) where $t_1 \models_M \phi \wedge AF(Fun, fair(Fun, \phi))$; due the same argument we can find states t_2, t_3 and so on. ■

The following proposition is a corollary from propositions 5 and 6.

Proposition 7. *Let* S *be a guarded scheme,* Fin *be a set of propositional variables,* M_S *be model constructed from* S *as specified in the definition 17, and* Fun *be the set of all program variables that correspond to propositional variables in* Fin *according to this definition. Then* S *is total with respect to* Fin *if and only if* $(0, m) \models_{M_S} fair(Fun, \bigvee_{p \in Fin} p)$ *for all* m *such that* $(0, m)$ *is a state in* D_S.

Definition 20. *Global model checking [2] for a program logic (a variant of* μ*-Calculus in particular) in a class of models for this logic is an algorithmic problem to compute (to construct) the set* $M(\phi)$ *for input model* M *(in this class) and input formula* ϕ *(of this logic).*

Proposition 8. *Generalized halting problem for guarded non-deterministic Yanov schemata is decidable in quadratic time* $O(n_A + n_C)^2$ *where* n_A *and* n_C *are numbers of assignments and choices in the input scheme.*

Proof. The upper bound $O(|\phi| \times |M|^{alt(\phi)})$ for model checking μC in finite models is well-known [2]; here

– $|\phi|$ is the total number of Boolean connectives and modalities in the input formula,
– $alt(\phi)$ is the maximal number of *alternations* of nesting μ/ν-constructs in the formula,
– $|M|$ is the overall size of the input model (i.e. the total number of states and edges).

Let S be a guarded scheme, Fin be a set of propositional variables, M_S be model constructed from S as specified in the definition 17. According to proposition 7, we have to model check formula $fair(Fun, \bigvee_{p \in Fin} p)$ in the finite model M_S. It suffices to remark that $|fair(Fun, \bigvee_{p \in Fin} p)|$ is some fixed constant, $alt(fair(Fun, \bigvee_{p \in Fin} p)) = 1$ and $|M_S| = O(n_A + n_C)^2$. ■

4 Main Results

4.1 Translation Algorithm

Let us define below a recursive algorithm $F2S$ (*Formulas To Schemata*) to translate normal guarded μS-formulas into guarded non-deterministic Yanov schemata. We would like to use in the definition the following standard control-flow constructs:

- $S'; S''$ for sequential composition of two schemata,
- $if\ q\ then\ S'\ else\ S''$ for deterministic choice in two schemata,
- $S' \cup S''$ for non-deterministic choice in two schemata;

all these control-flow constructs are easy to define formally in terms of non-deterministic Yanov schemata. Let us also use some macro-notations:

- *stop* for all final labels of a scheme under consideration (i.e. labels that occur in the scheme but don't mark any operator),
- *loop* for a fixed scheme that always loops (e.g., $\{0 :\ if\ p\ then\ \{0\}\ else\ \{0\}\}$).

Algorithm $F2S$:

- For any propositional variable
  - $F2S(p) = \{0 :\ if\ p\ then\ stop\ else\ loop\}$;
  - $F2S(\neg p) = \{0 :\ if\ p\ then\ loop\ else\ stop\}$;
- $F2S(\phi \wedge \psi) = if\ q\ then\ F2S(\phi)\ else\ F2S(\psi)$, where q is a new (fresh) propositional variable;
- $F2S(\phi \vee \psi) = F2S(\phi) \cup F2S(psi)$;
- for any program variable
$$F2S([a]\phi) = F2S(\langle a\rangle\phi) = \{0 :\ a\ goto\ stop\}; F2S(\phi).$$
- $F2S(\mu x.\phi)$ results from $F2S(\phi)$ by replacement instead of every choice operator (with condition x) "$if\ x\ then\ stop\ else\ loop$" the unconditional operator "$goto\ \{0\}$".
- $F2S(\nu x.\phi)$ results from $F2S(\phi)$ by replacement instead of every choice operator (with condition x) "$if\ x\ then\ stop\ else\ loop$" the choice operator "$if\ x\ then\ \{0\}\ else\ stop$".

Proposition 9. *Algorithm $F2S$ translates normal guarded μS-formulas into guarded non-deterministic Yanov schemata in linear time.*

Definition 21. *Let $M = (D, R, E)$ and $M' = (D', R', E')$ be two models with same domain (i.e. $D = D'$) and interpretation of program variables (i.e. $R = R'$), let Fin be a set of propositional variables; we say that M' is a* modification *of M on Fin, if valuations E and E' differs on variable in Fin only (i.e. $E(p) = E'(p)$ for every $p \notin Fin$).*

Proposition 10. *Let M be any Herbrand model, ϕ be a normal guarded formula of the strict μ-Calculus, Gfp (Greatest fix points) be the set of all propositional variables that are bound by ν in this formula, and Cnj be the set of all new*

propositional variables q that are introduce for conjunctions in an exercise of $F2S(\phi)$. Then μS-formula ϕ is valid in M if and only if the non-deterministic Yanov scheme $F2S(\phi)$ halts in every M' that fits Gfp and is a modification of M on $Gfp \cup Cnj$.

Proof. Induction on formula structure. Induction base is the case when a formula is a literal (i.e. a propositional variable or its negation); in this case proof is trivial due to explicit definition of $F2S$ in these cases.

Induction step in case of disjunction $\vee$ and modalities $[\ldots]$ or $\langle\ldots\rangle$ is straightforward due to simplicity of definition of $F2S$ in these cases.

Let us consider conjunction. Since $F2S(\phi \wedge \psi) = if\ q\ then\ F2S(\phi)\ else\ F2S(\psi)$, where q is a variable in Cnj, and since q has instances neither in $F2S(\phi)$ nor in $F2S(\psi)$, then we can interpret this variable arbitrary and (by this) test both $F2S(\phi)$ and $F2S(\psi)$ for halting.

Let us discuss an idea that is behind the induction step in case of $\mu p.\phi$. According to Corollary 3, $M(\mu x.\phi) = \bigcup_{n\geq 0} M(\phi^n_{false/x})$. Recall that $F2S(\mu x.\phi)$ results from $F2S(\phi)$ by substitution of unconditional "*goto* $\{0\}$" instead of "$if\ x\ then\ stop\ else\ loop$", i.e. $F2S(\mu x.\phi)$ is equivalent to $F2S(\bigvee_{n\geq 0} \phi^n_{false/x})$.

Finally, an idea behind induction step for $\nu p.\phi$ follows. Again, according to Corollary 3, $M(\nu x.\phi) = \bigcap_{n\geq 0} M(\phi^n_{true/x})$. Since $F2S(\nu x.\phi)$ results from $F2S(\phi)$ by substitution of "$if\ x\ then\ \{0\}\ else\ stop$" instead of "$if\ x\ then\ stop\ else\ loop$" (where $x \in Gfp$) then $F2S(\nu x.\phi)$ is equivalent to $F2S(\bigwedge_{n\geq 0} \phi^n_{true/x})$ because x can be true only finite number of times. ■

4.2 Results and Conclusion

Main Theorem follows from propositions 1-4 and 8-10.

Theorem 1. *The propositional μ-Calculus is decidable with exponential upper bound (on formula size).*

Since it is known that μ-Calculus is $EXP - Time$ complete [3], propositions 1-4 and 8-10 imply the following corollary.

Corollary 4.
Any algorithm that transforms μ-Calculus formulas into equivalent guarded formulas, must be exponential in fix-point nesting depth of the input formula.

Concluding Remarks. To the best of our knowledge, the lower bound from corollary 4 is a very new result [1]. Study of implications from this result (for parity games [15] for instance) may be a topic for further research. Another possible research topic may be complete axiomatization of the propositional μ-Calculus in a manner similar to the complete axiomatization for the Propositional Linear Temporal Logic in [12]. Research on a new approach to axiomatization may be interesting since completeness proof in [14, 15] uses reduction to guarded fragment.

References

1. Bruse F., Friedmann O., Lange M. Guarded Transformation for the Modal mu-Calculus. arXiv:1305.0648v2, 2013 (available at `http://arxiv.org/abs/1305.0648`).
2. Clarke E.M., Grumberg O., Peled D. Moedel Checking, MIT Press, 1999.
3. Emerson E.A. and Jutla C.J. The Complexity of Tree Automata and Logics of Programs. SIAM J. Comput., v. 29 (1999), 132-158.
4. Ershov A.P. Origins of programming: Discourses on metnodology, New York, Springer Verlag, 1990.
5. Harel D., Kozen D., Tiuryn J. Dynamic Logic, MIT Press, 2000.
6. Ianov Yu. I. The logical schemes of algorithms. In Problems of Cybernetics, v. I, A. A. Lyapunov, R Goodman, and A D. Booth (Eds), Pergamon Press, New York, 1960, 82-140.
7. Kotov V.E., Sabelfeld V.K. Theory of Program Schemata. Moscow, Nauka Publeshers, 1991. (In Russian.)
8. Kozen D. Results on the Propositional Mu-Calculus. Theoretical Computer Science, v.27, 1983, p.333-354.
9. Nepomniaschy V.A., Shilov N.V. Non-deterministic program schemata and their relation to dynamic logic. Internat. Conf. on Math. Logic and its Applications. Plenum Press, 1987, 137-147. (Revised version: Cybernetics, v.24(3) (1988), 285-293.)
10. Podlovchenko R.I. A.A. Lyapunov and A.P. Ershov in the Theory of Program Schemes and the Development of Its Logic Concepts. Perspectives of System Informatics, 4th International Andrei Ershov Memorial Conference, PSI 2001. Lecture Notes in Computer Science, v.2244, 2001, p.8-23.
11. Shilov N.V. Program Schemata vs. Automata for Decidability of Program Logics. Theor. Comput. Sci., v.175 (1997), 15-27.
12. Shilov N.V. An approach to design of automata-based axiomatization for propositional program and temporal logics (by example of linear temporal logic). In: Logic, Computation, Hierarchies. Series: Ontos Mathematical Logic, v.4 Ontos-Verlag/De Gruyter, Germany, 2014, p.297-324.
13. Tarski A. A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics, v.5, 1955, p.285-309.
14. Walukiewicz I. A Complete Deductive System for the mu-Calculus. Proceedings of IEEE LICS'93, 1993, p.136-147.
15. Walukiewicz I. Completeness of Kozen's axiomatisation of the propositional Mu-calculus. Information and Computation, v.157, 2000, p.142-182.