On the Formal Specification of Automata-based Programs via Specification Patterns

Andrey A. Klebanov

Abstract—Model checking is a well developed verification technique still it is not widely adopted. One of the reasons is that defining formal specification is an error-prone and time-consuming task. This paper gives an overview of the ongoing research which focuses on expressing verifiable requirements in controlled natural language in the framework of automata-based programming.

Index Terms—Model checking, Specification, Temporal logic, Language parsing and understanding.

1 Introduction

AUTOMATA-BASED programming [1] – is a software development paradigm based on the extended finite-state machine model. In this approach programs are represented as a system of automated controlled entities which behavior is described by the system of interacting state machines.

Model checking [2] could be successfully applied to automata-based programs [3], [4]. The idea of model checking is to verify the consistency between finite-state model (*Kripke structure*) and formal specification expressed as a set of temporal logic formulae. In verification the main advantage of automata-based approach over traditional ones is a high extent of automaticity as in automata-based programs behavior model is defined a priori. Several methods [5], [6], [7] have been developed to automatically transform both a control system into a verifiable model and a counter-example produced by a verification tool back into automata model. Still for all the approaches a significant obstacle exists – considerable mathematical background is required for expressing specification as a temporal logic formula.

Design by Contract approach [8] could partly solve this problem [9] as contracts are much simpler formalism. However they are second to temporal logics in expressive power – they can do no better than specifying invariance, precondition or postcondition properties.

This paper presents an approach which battles temporal logics' complexity. The requirements are expressed in a controlled natural language defined by a formal grammar introduced further. The grammar is based on a set of specification patterns [10], [11] – a generalized description (both formal and in natural language) of a commonly occurring requirement on a permissible state sequences in a finite-state model of a system. Thus for each requirement equivalent verifiable formal mapping could be defined.

In [3] specification patterns are mentioned in the framework of automata-based programming: "… it is important to consider temporal properties patterns (structures) which are most suitable and appropriate for automata-based programs verification. Existence of such patterns would allow focusing on classes of temporal properties of automata models which definitely would facilitate flow chart development for automata-based programs verification." Still only one requirement (which is an instance of existing pattern) is outlined and no further development is provided.

The rest of the paper is organized as follows. Section 2 covers some background material on the nature of specification patterns and how they can be adapted for automata-based programming. Section 3 discusses specification patterns applicability analysis results. Formal grammar to derive verifiable requirements is introduced in Section 4. Finally, Section 5 makes a conclusion.

2 Specification Patterns

Specification patterns system has been introduced in [10], [11]. They are based on a specifications analysis for the programs developed in a traditional (i.e. non automata-based) way.

Patterns could be classified according to the hierarchy based on their semantics. Eight patterns which belong to one of the groups ("Occurrence" and "Order") are outlined. Patterns which belong to the group "Occurrence" specify occurrence or absence of the states in which a given state formula holds. "Order" group contains patterns which describe order of the states during system execution.

Pattern is described by its name (or set of names), intent, mappings to some formalisms (LTL, CTL and etc.), example of use and relationships with other patterns.

Each requirement has a scope – an extent of the system execution over which it should hold. Five kinds of scopes are defined:

- Global – entire execution path.

The research is conducted in scope of the Federal target program "Scientific and pedagogical personnel of innovative Russia for 2009 – 2013 years".

- *A. A. Klebanov is with the Computer Technologies Department, Saint-Petersburg State University of Information Technologies, Mechanics and Optics (SPbSU IFMO), Saint-Petersburg, Russia. E-mail: klebanov.andrey@gmail.com.*
- *The research is supervised by O. G. Stepanov, PhD, who is with the Computer Technologies Department, SPbSU IFMO, Saint-Petersburg, Russia.*

TABLE 1
"UNIVERSALITY" PATTERN

Intent		The pattern is used to describe a portion of a system's execution which contains *only* states that have a desired property. Also known as "Henceforth" and "Always".	
Mapping	LTL	Scope	Mapping
		Globally	□(P)
		Before R	◊R → (P U R)
		After Q	□(Q → □(P))
		Between Q and R	□((Q & !R & ◊R) → (P U R))
		After Q until R	□(Q & !R → (P W R))
	CTL	Scope	Mapping
		Globally	AG(P)
		Before R	A[(P | AG(!R)) W R]
		After Q	AG(Q → AG(P))
		Between Q and R	AG(Q & !R → A[(P | AG(!R)) W R])
		After Q until R	AG(Q & !R → A[P W R])
Example and known uses		The pattern could be used to specify either entire model or some group of states properties. For example when it's desired to express requirement like: "If an automaton is in state s, then P holds."	
Relationships with other patterns		The pattern is closely related to the "Absence" and "Existence" patterns. Universality of a state can be viewed as absence of its negation.	

- Before - execution path up to a given state.
- After – execution path after a given state.
- Between – execution between two given states.
- After-until – the same as "Between", but the right end of the interval where property holds is optional.

For the state-oriented formalism intervals are left-closed and right-open.

As an example, "Universality" pattern is presented in Table 1. Original "Example and known uses" section's contents is substituted with an automata-oriented example, this is a key idea behind patterns adaption for automata-based paradigm.

3 SPECIFICATION PATTERNS APPLICABILITY ANALYSIS

Creating a new pattern system for the formal specification of automata-based programs wouldn't make much sense without preliminary applicability analysis of the specification patterns described before. To carry it out it's necessary to analyze how requirements for automata-based programs (developed in SPbSU IFMO, Yaroslavl State University, Concern AVRORA and available at [12]) could be expressed via specification patterns. An example of intermediate results organization is presented in Table 2. Columns "Requirement" and "Original formal mapping" represent original requirements from the source (column "Source") expressed in natural language and one of the formalisms correspondingly. Pattern which instantiation with a real requirement leads to a formal equivalent is provided in column "Pattern, Scope". Equivalence proof is provided where required.

TABLE 2
INTERMEDIATE ANALYSIS EXAMPLE

Requirement	Original formal mapping	Pattern, Scope	Source
If either heater of one of the valves failure has happened, then coffee machine (automaton A_0) will mandatory change its state to the state 5.	AG((y_{31} = 4 | y_{32} = 4 | y_2 = 4) & y_0 = 2 → A(y_0 = 2 U y_0 = 5)))	Response (constrained), Globally AG(P → A(S)), P: (y_{31} = 4 | y_{32} = 4 | y_2 = 4) & y_0 = 2, S: y_0 = 2 U y_0 = 5	[4]

Altogether 77 requirements for 13 programs from 15 sources have been analyzed. 87% of the requirements could be expressed via five patterns. Remaining 13% couldn't be expressed due to the limitations of the pattern system or issues of the concrete automata-based model of the system. The percentage between used patterns is presented on the Fig. 1 and between used scopes – on the Fig. 2.

4 CONTROLLED NATURAL LANGUAGE GRAMMAR

Several approaches to extract verifiable requirements form the natural language specifications have been developed. Among the most popular [13] are natural language processing and formal grammars-based derivations.

In this paper grammar-oriented approach is used. The grammar is based on the specification patterns. Also wide spread variants of some patterns (i.e. "Response" and chain patterns) are added explicitly. So the requirement could be expressed both in natural language and in any formalism supported by the pattern system. An extract of the grammar is presented in Table 3; placeholders for real requirements are in `monospace font`.

To exemplify this, requirement from [4] is considered: "Coffee machine control system never gets to the state where it doesn't respond to either system timer events, or buttons "OK" or "Cancel". In the automata-based model of the coffee machine control system requirement "Doesn't respond to either system timer events, or buttons "OK" or "Cancel" corresponds to the predicate `act = end`. The adverb "never" implies to use "Absence" pat-

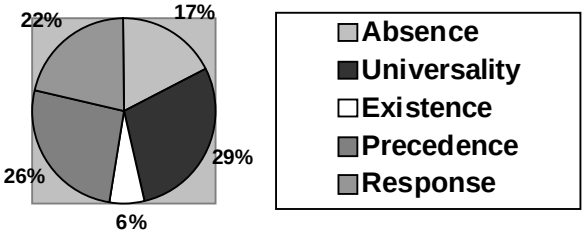

Fig. 1. The percentage between used patterns.

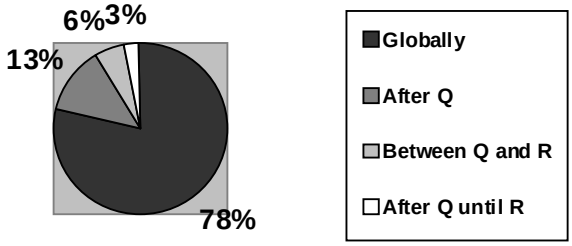

Fig. 2. The percentage between used scopes.

tern with a "Global" scope. The derivation follows:

<requirement> → <scope> <pattern> → For all the states holds <pattern> → For all the states holds <absence> → For all the states holds that never P.

Instantiating `P` with a real requirement leads to a desired requirement in natural language: "For all the states holds that never `act = end`." Formal expressions in CTL and LTL used for verification purposes are `AG(!act = end)` and `□(!act = end)` correspondingly.

5 Conclusion

Requirements expression as temporal logic formulae is error-prone and time-consuming task. An approach which facilitates this process has been introduced in this paper.

There are a few open issues left to work on in future. First of all it's tool support. It has been shown in [9] how JetBrains Meta Programming System [14] could be used both to develop and verify automata-based programs. Currently formal specifications are integrated with code but only as temporal logic formulae. The major improvement would be to replace them with the natural language specifications as described above. Besides similar to [13], [15], [16] some wizard to guide a user during property construction could be implemented. Finally, further use cases of the pattern system could be investigated.

References

[1] N.I. Polikarpova, A.A. Shalyto, *Automata-based programming*, Piter, 2009. (in Russian)

[2] E.M. Clarke, O. Grumberg, D.A. Peled, *Model Checking*, MIT Press, 2000.

[3] K.A. Vasileva, E.V. Kuzmin, "LTL Verification of Automaton Programs," *Modeling and Analysis of Information Systems*, vol. 14, no. 1, pp. 3–14, 2007. (in Russian)

[4] E.V. Kuzmin, V.A. Sokolov, "Modeling, Specification, and Verification of Automaton Programs," *Programming and Computer Software*, vol. 34, no. 1, pp. 38–60, 2008. (in Russian)

TABLE 3
CONTROLLED LANGUAGE GRAMMAR EXTRACT

<requirement>	::= <scope> <pattern>
<scope>	::= «For all the states holds that» | «Before the state where Q, holds that» | «After the state where Q, holds that» | «Between the states where Q and R, holds that» | «After the state where Q, before the state where R, holds that»
<pattern>	::= <absence> | <universality> | <existence> | <constrained existence> | <precedence> | <response> | <constrained response> | <chain precedence> …
<absence>	::= «never P.»
…	…
<response>	::= «always if P, then eventually S.»
…	…

[5] V.S. Gurov, B.R. Yaminov, "*Automata-based Programs Verification without Translation into Verification Tool's Input Language*," Proc. Conf. Scientific Software in Education and Research. 2008. (in Russian)

[6] M.A. Lukin, A.A. Shalyto, "*Automation of Visual Automata-based Programs Verification*," Proc. 15th Int'l. Conf. Advanced Intellectual Technologies and Innovation in Education and Science. 2008. (in Russian)

[7] E. Kurbatsky, "*Verification of Automata-Based Programs*," Proc. Sec. Spring Young Researchers Colloquium Software Engineering. 2008.

[8] B. Meyer, *Object-Oriented Software Construction, 2nd Edition*, Prentice Hall PTR, 2000.

[9] A. Borisenko, P. Fedotov, O. Stepanov, A. Shalyto, "*Reliable Software with Complex Behavior Development*," Proc. 5th Central and Eastern European Software Engineering Conf. in Russia. 2009.

[10] M.B. Dwyer, G.S. Avrunin, J.C. Corbett, "*Property Specification Patterns for Finite-state Verification*," Proc. 2nd Workshop Formal Methods in Software Practice. 1998.

[11] M.B. Dwyer, G.S. Avrunin, J.C. Corbett, "*Patterns in Property Specifications for Finite-state Verification*," Proc. 21st Int'l. Conf. Software Engineering. 1999.

[12] *Programming Technologies Department*, *Saint Petersburg State University of Information Technologies, Mechanics and Optics*, http://is.ifmo.ru/

[13] S. Konrad, B.H.C. Cheng, "*Facilitating the Construction of Specification Pattern-based Properties*," Proc. IEEE Int'l. Requirements Engineering Conf. 2005.

[14] *JetBrains Meta Programming System*, http://www.jetbrains.com/mps/index.html

[15] R.L. Smith, G.S. Avrunin, L.A. Clarke, L.J. Osterweil, "*PROPEL: An Approach Supporting Property Elucidation*," Proc. 24th Int'l. Conf. Software Engineering. 2002.

[16] O. Mondragon, A.Q. Gates, S. Roach, "*Prospec: Support for Elicitation and Formal Specification of Software Properties*," Proc. Runtime Verification Workshop. 2004.