ISSN 1818–1015 (Print)
ISSN 2313–5417 (Online)

Министерство образования и науки Российской Федерации
Ярославский государственный университет им. П. Г. Демидова

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ

Том 25 № 5 (77) 2018

Основан в 1999 году
Выходит 6 раз в год

Адрес редакции: ЯрГУ, ул. Советская, 14, г. Ярославль, 150003, Россия
Website: http://mais-journal.ru, e-mail: mais@uniyar.ac.ru; телефон (4852) 79-77-73

Научные статьи в журнал принимаются по электронной почте. Статьи должны содержать УДК, аннотации на русском и английском языках и сопровождаться набором текста в редакторе LaTeX. Плата с аспирантов за публикацию рукописей не взимается.

12+

СОДЕРЖАНИЕ

Моделирование и анализ информационных систем. Т. 25, №5. 2018

Свидетельство о регистрации СМИ ПИ № ФС 77 – 66186 от 20.06.2016 выдано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций. Учредитель – Федеральное государственное бюджетное образовательное учреждение высшего образования "Ярославский государственный университет им. П. Г. Демидова". Подписной индекс – 31907 в Объединенном каталоге "Пресса России". Редактор, корректор А.А. Аладьева. Редактор перевода Э.И. Соколова. Подписано в печать 17.10.2018. Дата выхода в свет 31.10.2018. Формат 60x84$^1/_8$. Усл. печ. л. 14,6. Уч.-изд. л. 13,0. Объем 125 с. Тираж 46 экз. Свободная цена. Заказ 101/018. Адрес типографии: ул. Советская, 14, оф. 109, г. Ярославль, 150003 Россия. Адрес издателя: Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, г. Ярославль, 150003 Россия.

ISSN 1818–1015 (Print)
ISSN 2313–5417 (Online)

P.G. Demidov Yaroslavl State University

MODELING AND ANALYSIS OF INFORMATION SYSTEMS

Volume 25 No 5 (77) 2018

Founded in 1999
6 issues per year

Editorial Office Address: P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia
Website: http://mais-journal.ru, e-mail: mais@uniyar.ac.ru

Contents

Modeling and Analysis of Information Systems. Vol. 25, No 5. 2018

Program Verification

Applied Logics

Program Synthesis and Transformations

Automata Theory

Neural Network Dynamics

От редакторов специального выпуска

В. А. Захаров, Н. В. Шилов

21–22 июня 2018 г. в Ярославле прошел девятый международный научно-исследовательский семинар «Семантика, спецификация и верификация программ: теория и приложения» (9-th Workshop on Program Semantics, Specification and Verification: Theory and Applications, PSSV 2018). Организатором семинара выступил факультет информатики и вычислительной техники Ярославского государственного университета имени П.Г. Демидова. Программа семинара PSSV 2018 включала 8 регулярных докладов, 3 лекции приглашенных докладчиков, а также мемориальную сессию, посвященную памяти Бориса Абрамовича Трахтенброта (1921–2016), Михаила Иосифовича Дехтяря (1946–2018) и Марса Котдусовича Валиева (1942–2018). В выступлениях участников семинара были представлены результаты завершенных и продолжающихся исследований разнообразных задач в области математического моделирования, верификации программных систем, прикладных логик, автоматизации логического вывода, теории автоматов. Значительное внимание было уделено методам дедуктивной проверки правильности программ, верификации моделей программ, применению методов теории автоматов к тестированию программ, а также ряду вопросов построения и анализа формальных моделей информационных систем. Данный выпуск журнала включает 5 статей участников семинара PSSV 2018.

22–25 мая 2018 г. в доме отдыха МГУ «Красновидово» прошла десятая международная конференция «Дискретные модели в теории управляющих систем» (ДМТУС 2018). Организатором конференции выступил факультет ВМК Московского государственного университета имени М.В. Ломоносова. Работа конференции проводилась по трем секциям и включала 8 пленарных и 75 секционных докладов. Одна из секций конференции была посвящена дискретным моделям в задачах программирования, искусственного интеллекта и теории управления. Избранные доклады участников этой секции были рекомендованы программным комитетом конференции для публикации в журнале «Моделирование и анализ информационных систем». Данный выпуск журнала включает 2 статьи участников конференции «Дискретные модели в теории управляющих систем».

В статье Т. Баара и С.М. Старолетова описан метод верификации гибридных программ с использованием интерактивного средства доказательства теорем KeYmaera. Авторы предлагают использовать графический язык представления анализируемых программ и проводить разметку состояний граф-схемы программы инвариантами и контрактами. За счет такой разметки оказывается возможным получать гораздо более простые доказательства. Описание предложенного метода дедуктивной верификации иллюстрируется примерами. Статья написана по материалам доклада, представленного на PSSV 2018.

В статье Х.Л. Вега Висе и В.Ю. Михайлова исследуется задача анализа программ развития сельскохозяйственных территорий. Для ее решения применяется аппарат описания и анализа формальных систем. В работе предлагается аксиоматическая семантика программ развития, опирающаяся на логику Хоара. Для программы развития ставится задача проверки достижимости заданного конечного набора значений переменных из заданного начального. Описан метод решения этой задачи с использованием языка PDDL. Статья написана по материалам доклада, представленного на ДМТУС 2018.

Статья Д.А. Кондратьева, И.В. Марьясова и В.А. Непомнящего посвящена развитию методов верификации циклов специфического вида на языке C, не требующих аннотации кода инвариантами и оценочными функциями. Представленная техника расширяет метод, описанный авторами в ранее опубликованных работах по этой теме на случай циклов, опе-

рирующих изменяемыми структурами данных. Статья написана по материалам доклада, представленного на PSSV 2018.

В статье А.Р. Гнатенко и В.А. Захарова проводится сравнительный анализ выразительных возможностей некоторых расширений пропозициональной логики предикатов линейного времени с другими логиками, используемыми для спецификации поведения реагирующих систем. Статья написана по материалам доклада, представленного на PSSV 2018.

С.М. Дудаков представил в своей статье новые результаты исследования вопроса о существовании т.н. инфляционной неподвижной точки (IFP), вычисляемой итеративной процедурой для операторов, выраженных формулами логики предикатов и соответствующих рекурсивным SQL запросам, в заданных алгебраических системах (универсумах). Статья написана по материалам доклада, представленного на PSSV 2018.

В статье С.А. Гречаника введено понятие полипрограммы в качестве объекта формализации и эквивалентных преобразований, исследуемых автором, и описана система эквивалентных преобразований полипрограмм, состоящая из двух частей: локальные правила и бисимуляция. Основное внимание уделено исследованию отношения бисимуляции полипрограмм. Статья написана по материалам доклада, представленного на PSSV 2018.

Н.В. Шилов в своей статье вновь привлекает внимание к вопросам о том, насколько глубоко эквивалентные преобразования могут затрагивать устройство программ и улучшать их производительность и какую роль в этом играют специальные свойства тех базовых операций и отношений, которые используются в программе. Задача, рассматриваемая в этой статье, заслуживает того, чтобы с ней ознакомились исследователи, заинтересованные в изучении вопросов выявления общих принципов построения оптимальных преобразований программ.

В статье Е.В. Хворостухиной и В.А. Молчанова исследуются некоторые свойства гиперграфических автоматов, в которых в качестве множеств состояний и выходных элементов используются гиперграфы. Эта разновидность автоматов включает в себя многие ранее известные и изученные классы автоматов, работающих над сложными структурами данных, и может быть использована в качестве математической модели некоторых классов информационных систем. В статье рассматривается задача выбора подходящего представления для универсального гиперграфического автомата, работающего над одним классом гиперграфов. Статья написана по материалам доклада, представленного на ДМТУС 2018.

В заключительной статье С.Д. Глызина и Е.А. Марушкиной рассматривается модель нейронной ассоциации из трех импульсных нейронов с вещательной запаздывающей связью между ними. На основе локальных асимптотических и связанных с ними численных методов изучена периодическая, квазипериодическая и хаотическая динамика этой сети.

Моделирование и анализ информационных систем. Т. 25, № 5 (2018), с. 465–480
Modeling and Analysis of Information Systems. Vol. 25, No 5 (2018), pp. 465–480

Верификация программ
Program Verification

DOI: 10.18255/1818-1015-2018-5-465-480

UDC 004.942

A Control Flow Graph Based Approach to Make the Verification of Cyber-Physical Systems Using KeYmaera Easier

Baar T., Staroletov S.

Received September 10, 2018

Abstract. KeYmaera is an interactive theorem prover and is used to verify safety properties of cyber-physical systems (CPSs). It implements a Dynamic Logic for Hybrid Programs (HPs), while a HP models a CPS very precisely. Verifying properties of a given system in KeYmaera can become a challenge for a user since the proof is authored in a classical sequent calculus framework and a successful proof requires from the user intimate knowledge of the available calculus rules. Another barrier for widespread application of KeYmaera is the purely textual representation of current proof goals, what requires from the user very good training, experience, and patience.

In this paper, we present an alternative verification approach based on KeYmaera, which drastically improves usability and minimizes user interaction. The main idea is to let the user annotate invariants and contracts to states of the hybrid automaton.

Thus, the user can employ the graphical representation of the modelled system and is not bound to the purely textual form of hybrid programs as in KeYmaera. Based on the user-provided contracts, one can generate proof obligations, which are much simpler than the original proof goal in KeYmaera.

The article is published in the authors' wording.

Keywords: CPS, KeYmaera, proof contracts, verification, hybrid systems, usability, interactive provers

For citation: Baar T., Staroletov S., "A Control Flow Graph Based Approach to Make the Verification of Cyber-Physical Systems Using KeYmaera Easier", *Modeling and Analysis of Information Systems*, **25**:5 (2018), 465–480.

On the authors:
Thomas Baar, orcid.org/0000-0002-8443-1558, PhD,
Hochschule für Technik und Wirtschaft Berlin University of Applied Sciences, Germany
75 A Wilhelminenhofstrasse, D-12459, Berlin, Germany, e-mail: thomas.baar@htw-berlin.de

Sergey Staroletov, orcid.org/0000-0001-5183-9736, PhD,
Polzunov Altai State Technical University,
46 Lenina avenue, Barnaul, Altai region, 656038 Russian Federation, e-mail: serg_soft@mail.ru

1. Motivation

A cyber-physical system (CPS) is a system that tightly combines software with physical components. The state of a CPS consists of the discrete state of its software and the analogous state of its physical parts. Safety analysis of CPSs must take into account

physical laws, which apply to physical parts as well as the code structure of the software part [8].

The notion of hybrid automaton (HA) [3, 9] has proven to be useful for the precise description of the behaviour of CPSs. Like a classical UML state machine [5], a hybrid automaton consists of states, transitions between states, and state variables. Transitions can carry annotations for both an execution condition and an action. An action changes the value of a state variable upon executing the transition. New in hybrid automata is, that the value of (some) state variables (called *continuous state variables*) can change according to given differential equations when the system has entered a long-running state. This extension of hybrid automata to classical UML state machines reflects the physical parts of the modeled system. For examples, the current position (z) of the system changes according to the current velocity by $z' = v$, where z' denotes the derivation of z over time.

Logic-based analysis of a given hybrid automaton has been thoroughly investigated by Platzer in [12] and became practically feasible by the tool KeYmaera [14]. This tool is an interactive theorem prover and allows the user to formally prove safety properties taken both discrete and continuous state variables into account. However, KeYmaera does not work directly on the hybrid automaton but needs as input a so-called *hybrid program* (HP). A hybrid automaton can be seen as the control flow graph of a hybrid program.

In this paper, we propose an approach to overcome some of the obstacles the user faces when authoring a proof using KeYmaera. One enormous problem is the complexity of proofs due to the length and complexity of the system implementation represented by a hybrid program. KeYmaera expects a proof goal of form $preCond \rightarrow [\alpha]postCond$, where α is the hybrid program representing the *whole* system implementation.

Instead, our approach applies the idea already formulated in 1967 by Robert W. Floyd [7] for flowchart verification on the verification of a CPS: The user is allowed to annotate the control flow graph of hybrid program α with fine grained knowledge about intermediate states. This additional knowledge can be given in form of invariants (similar to loop invariants) and contracts for long-running states. Based on the provided invariants and contracts, one can generate proof obligations, which are much simpler than the original proof goal in KeYmaera and can often be automatically discarded.

2. Verification of CPSs using KeYmaera

In KeYmaera, a CPS is modelled in form of a Hybrid Program (HP), for which properties expressed in Dynamic Logic can be proven. A HP is built on variables (always of type $float$), derivations of (continuous) variables, arithmetic expressions, first-order formulas for conditions on the current state, and a simple execution language with operators for assignment ($:=$), sequential execution ($;$), non-deterministic repetition ($*$), and others. For a detailed introduction to HP and the logic of KeYmaera, the reader is referred to [13].

2.1. Running Example: Simple Velocity Controller

As an illustrative example, we introduce a simple velocity controller. The velocity v of the controlled system (e.g. a car or a train) is set by the controller either to a fixed velocity v_0 or to 0 (zero). Note that the controlled system is moving if $v > 0$, i.e. the system's position (encoded by z) changes for a time-period Δ with $z = z + v * \Delta$. The change of the system's position based on the current velocity v is a *physical law*, which holds independently from the considered controller and has to be taken into account for all long-running states the systems can be in. An alternative (and widely-established) notation for this law is $z' = v$, what is more general than the above $z = z + v * \Delta$, since velocity v might now even change over time.

Our simple velocity controller periodically updates the chosen velocity based on the information how far away from an obstacle (whose position is encoded with m) the system currently is. If the distance to the obstacle $m - z$ is greater than what the system can move within a period ϵ (encoded in our program as variable SB), the system will keep velocity $v = v_0$. Otherwise controller sets $v = 0$, what means that the system stops (very abruptly). The safety property we want to prove is, that the controller never stops the system too late, i.e. under all circumstances we will have $z < m$.

Our example is actually a simplified version of the tutorial example given in [13] and formulated as a Hybrid Program α as follows:

$$
\begin{array}{ll}
 & \{ \\
 & \quad SB := m - \epsilon * v_0; \\
 & \quad if \quad z < SB \\
\alpha = & \quad then \quad \{v := v_0; t := 0; z' = v, t' = 1 \,\&\, t \leq \epsilon\} \\
 & \quad else \quad \{v := 0; z' = v\} \\
 & \quad endif \\
 & \}*
\end{array}
$$

The Hybrid Program α has the form of the nondeterministic repetition $*$ of a block $\{\ldots\}$ while within the block we have a sequence of statements (separated by ;). Nondeterministic repetition means, that the annotated block can be executed arbitrarily often, including 0 times.

Inside the block, the first statement is the assignment $SB := m - \epsilon * v_0$. The second (and last) statement in the block is an $if - then - else$ statement.

The then-branch is a block consisting of assignments $v := v_0$ and $t := 0$ followed by the last statement in the block: a reference to a long-running state with properties $z' = v, t' = 1 \,\&\, t \leq \epsilon$. Note that after entering a long-running state, the system can remain in this state as long as the state's *domain constraint* (here $t \leq \epsilon$) permits. However, the system can leave the long-running state at any time (nondeterministically) and the program α proceeds with executing the next statement. The differential equation $z' = v$ is the physical law already discussed above while $t' = 1$ is a helper equation for a new variable t. Each long-running state can be annotated with an already mentioned domain constraint, which indicates conditions that must hold as long as the systems stays in this long-running state. In other words: The system must leave the long-running state at latest when the domain constraints flips from $true$ to $false$. In our case, the domain constraint is $t \leq \epsilon$. Together with the equation $t' = 1$ the domain constraint

ensures, that the system stays a maximum period of ϵ in this long-running state.

The else-branch $v := 0; z' = v$ is similar to the then-branch and consists of an assignment and a long-running state.

In the remaining paper, we would like to prove for our hybrid program α the safety property that the system will never reach the obstacle at position m, when it is started in a position smaller than m. Formally, this safety property reads as:

$$z < m \wedge \epsilon > 0 \wedge v_0 > 0 \rightarrow [\alpha] z < m$$

3. Our Approach: Graphical Representation and Contracts

Starting with the textual hybrid program α shown above, we extract the control flow graph (CFG) of α as shown in Fig. 1. The only difference to the original definition of α, that the two long-running states in the program are now named with *driving* and *stopped*.

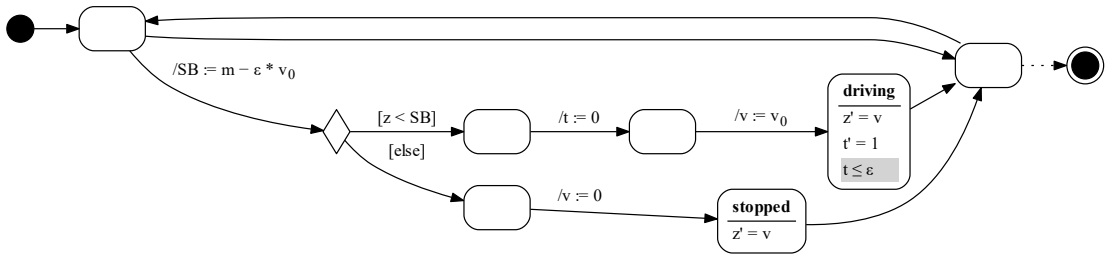


Fig. 1. Control Flow Graph for α (Hybrid Automaton)

The program α is executed by entering the graph via the start node (left side) and following the transitions between the nodes. Transitions can be annotated with an assignment (e.g. $SB := m - \epsilon * v_0$) or with a condition (e.g. $z < SB$). The diamond in the graph represents an if-then-else statement. The nodes for the long-running state contain the annotated differential equations and the domain constraint (gray background). Note that our CFG is very close to the notation of Hybrid Automata [9].

In a second step, our diagram is extended with the safety property to be proven as shown in Fig. 2. This diagram contains notes for a pre- and a post-condition. The tool KeYmaera is supposed to prove now $pre \rightarrow [\alpha] post$, but due to the complexity of α, it often becomes a challenge to manually create a proof using KeYmaera for this claim.

In a third step, we want to make the work of the KeYmaera user much easier. The idea is to provide a system description that already contains key facts for proving the correctness as shown in Fig. 3.

We allow the user to add so-called *proof contracts*. A proof contract can be a pre-/post-condition attached to a long-running state or an invariant attached to a normal state. For example, the desired property $z < m$ basically holds in every state of the

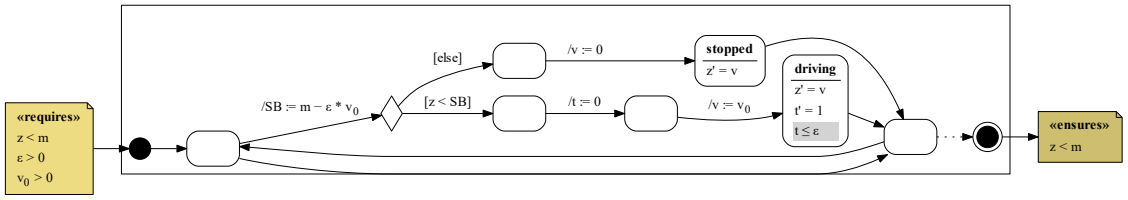


Fig. 2. System specification for α including pre-/post-condition

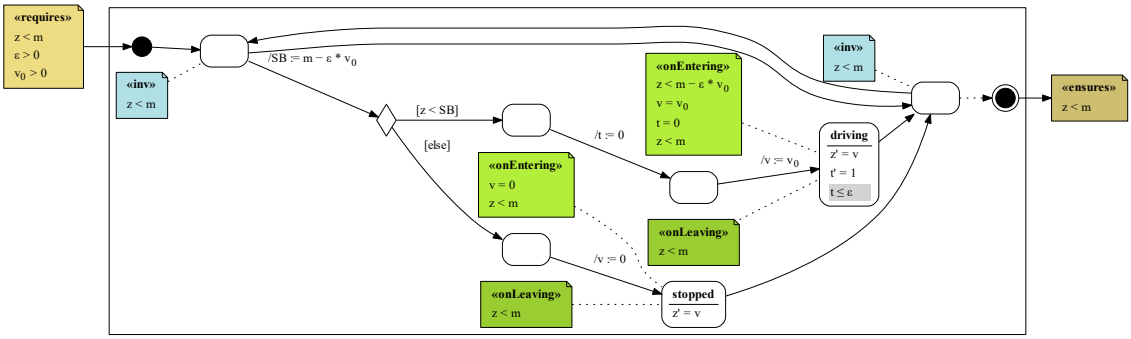


Fig. 3. Proof contracts have been added

control flow graph and especially in the nodes after the start state and before the end state (indicated in Fig. 3 by notes with stereotype ≪inv≫). The long-running state *driving* has now attached a pre-condition $z < m - \epsilon * v_0 \wedge v = v_0 \wedge z < m$ (indicated by note with stereotype ≪onEntering≫; the lines within the note are implicitly connected with logical conjunction $\wedge$) and the post-condition $z < m$ (indicated by a note with stereotype ≪onLeaving≫).

3.1. Generation of Proof Obligations for the Control Flow Graph

Once we have annotated the control flow graph with additional proof contracts, we want to know whether the annotated graph is still correct. Correctness basically means, that between any annotated states s_{pre} and s_{post}, which are connected by a *transition path* $t_1, \ldots, t_n$, the property specified for s_{post} holds, whenever the system evolves from state s_{pre} with its specified properties and executes transitions $t_1, \ldots, t_n$.

To illustrate the approach, all transition paths resulting into proof obligations are marked in Fig. 4. The proof obligations can be grouped according to their characteristics. We discuss here only the most important aspects of the proof obligations. The full version of all proof obligations can be found in appendix A. They are written in the input syntax of KeYmaera.

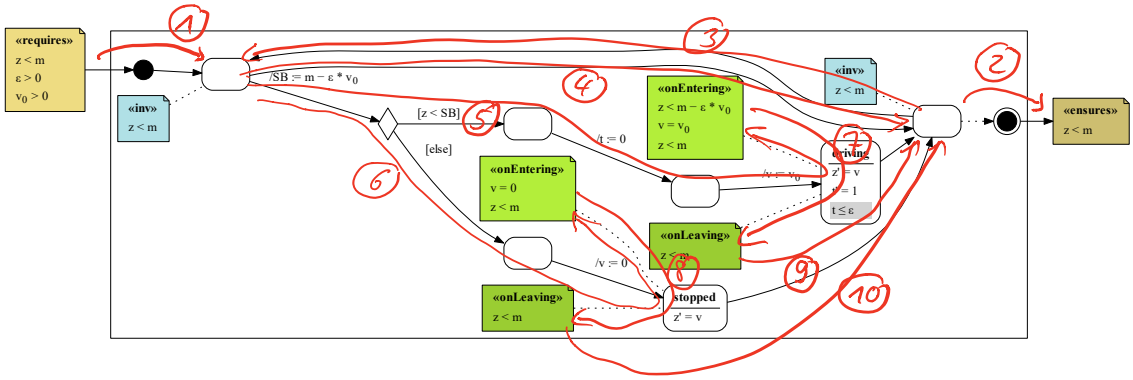


Fig. 4. Transition paths resulting into proof obligations

Transition paths between ≪requires≫/≪ensures≫ properties and first/last state of the system (①, ②)

The control flow graph starts always with a start node and finishes with a final node. The start/final node has a successor/predecessor node respectively, which are annotated with an invariant (in our case, the invariant is in both cases $z < m$, cmp. Fig. 4). Two proof obligations must now ensure, that

- the ≪requires≫ properties entails the invariant of the first node (i.e., the successor node of the start state) – see transition path ①
- the ≪ensures≫ properties is entailed by the invariant of the last node (i.e., the predecessor node of the final state) – see transition path ②

In mathematical notation we have:
$REQUIRES \rightarrow INV_{first}$
$INV_{last} \rightarrow ENSURES$

Transitions paths between invariant nodes (③, ④)

There are two nodes in the system that are annotated with an invariant (called the first/last node, see above). Both nodes are connected by a direct transition in each direction. In our case, these two transitions connect the two states directly and they do not have any annotation (no condition, no assignment). However, annotations would be allowed as well as a sequence of transitions to establish the connection between two invariant nodes.

If two invariant nodes n_1, n_2 are connected with a transition path $t_1, \ldots, t_n$, then the proof obligation has to ensure that the invariant of n_2 is entailed by invariant of n_1 followed by the execution of $\{t_1; \ldots; t_n\}$

In mathematical notation we have:
$INV_{n_1} \rightarrow [\{t_1; \ldots; t_n\}] INV_{n_2}$

Transitions paths between invariant node and ≪onEntering≫ property (⑤, ⑥)

There might be also a transition path $t_1, \ldots, t_n$ connecting an invariant node n_1 with a long-running state n_2. The long-running state must be annotated with an ≪onEntering≫ property. The proof obligation has to ensure that the ≪onEntering≫ property is entailed by invariant of n_1 followed by the execution of $\{t_1; \ldots; t_n\}$

In mathematical notation we have:

$INV_{n_1} \rightarrow [\{t_1; \ldots; t_n\}] ONENTERING_{n_2}$

Note that this is the first proof obligation in our example, in which $\{t_1; \ldots; t_n\}$ is not an empty sequence since the transitions are really annotated with condition/assignment (cmp. appendix A).

Relating ≪onEntering≫ and ≪onLeaving≫ properties for each long-running state (⑦, ⑧)

For each long-running state n, there is a proof obligation that the ≪onLeaving≫ property is entailed by the given ≪onEntering≫ property and the differential equations including the domain constraint attached to the long-running state. Proving this entailment might be non-trivial and usually needs some additional help from the user.

Let's consider node *stopped* for a rather simple example. We have to prove that from the ≪onEntering≫ property $v = 0 \wedge z < m$ and the differential equation $z' = v$ the ≪onLeaving≫ property $z < m$ follows. However, the value of variable z in the ≪onLeaving≫ property might be different from the value for z in the ≪onEntering≫ property. Furthermore, we have to deal with the differential equation $z' = v$, for which our formalism for proof obligations (first-order logic) is not made for.

To cope with this problem, we introduce separate versions of the variables in the ≪onLeaving≫ property and substitute the original variables with the new version. For example, the ≪onLeaving≫ property $z < m$ become $z_{out} < m$ when we introduce z_{out} for z. In addition, we let the user formulate additional formulas to resolve the differential equations attached to the long-running state. In case of state *stopped*, the user might resolve $z' = v$ to $z_{out} = z + v * \Delta$, where Δ encodes the time the systems stays in the long-running state *stopped*. Sometimes, it is important to know that $\Delta >= 0$ holds.

For the long-running state *stopped*, we come up with the following proof obligations: $(v = 0 \wedge z < m) \wedge z_{out} = z + v * \Delta \wedge 0 <= \Delta \rightarrow z_{out} < m$

In mathematical notation we have:

$(ONENTERING_n \wedge AXIOMS\_FOR\_DIFFEQS \wedge$

$DOMAIN\_CONSTRAINT_n \wedge DOMAIN\_CONSTRAINT_n[v \leftarrow v_{out}])$

$\rightarrow ONLEAVING_n[v \leftarrow v_{out}]$

for all variables v that might change their value in state n. Note that $f[v \leftarrow v_{out}]$ denotes the substitution of v by v_{out} in f.

Transitions paths between «onLeaving» property and invariant node (⑨, ⑩)

If a long-running node n_1 is succeeded by a transition path $t_1, \ldots, t_n$ going to an invariant node n_2, we need a proof obligation showing that after leaving n_1 and executing the transition path $t_1, \ldots, t_n$, the invariant of n_2 is entailed.

In mathematical notation we have:
$ONLEAVING_{n_1} \rightarrow [\{t_1; \ldots; t_n\}]INV_{n_2}$

3.2. Discarding the generated proof obligations using KeYmaera

We used KeYmaera version 3.6.17 to show the validity of generated proof obligations. Our KeYmaera installation was 'pure' in the sense that no background prover such as Reduce, Z3, or Mathematica was configured.

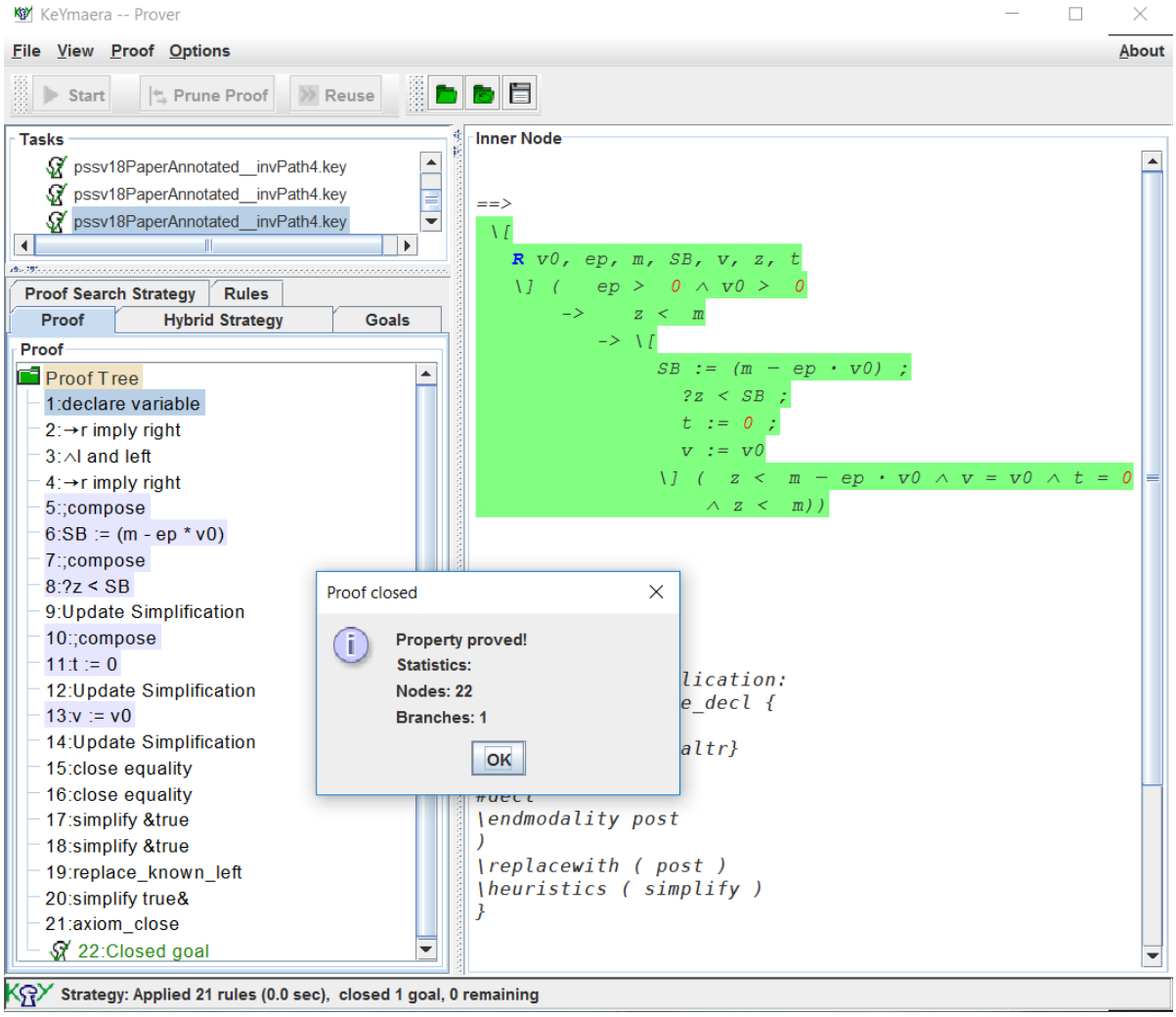


Fig. 5. Automatic proof of obligation using KeYmaera

Nevertheless, all generated obligations except of one could be proven automatically without any additional user interaction. Fig. 5 shows as a typical example the automatic proof of the non-trivial obligation ⑤. The only proof obligation that could not be discarded automatically was obligation ⑦, because the proof exploits transitivity of the $<$ relationship. Nonetheless, the prover PRINCESS [15] could prove also this obligation fully automatically.

4. Related Work

According to the nature of CPSs, we can identify three main layers for their specification: (1) transition automata with discrete jumps, (2) continuous dynamic calculations in each state (what makes the system to be a hybrid system) and (3) safety properties which are interesting for a user to check.

In this section we browse among some known techniques to verify such specifications and describe issues one can face. We use a simple demo system with the velocity controller and the simple goal $z < m$.

Firstly, the user's goals about constant or unexpected behaviour can be easily transformed into LTL(Linear Time Logic) formulas with temporal operators "always" and "eventually". For example, if the property $z < m$ is supposed to hold in all reachable states during execution of the CPS, then we can express this as $[]z < m$.

But after expressing the goals we will get a major issue when we try to describe the behaviour model of a system: It is near to impossible to implement the continuous (or at least close to continuous) dynamic behaviour in each state, even if we hard code the mathematical expressions and solve it without loose of accuracy. The main problem here is an explosion of the number of internal states and memory being used in a verifier to express such a system.

According to the design of a well-known tool in Model Based Checking world, Spin's Promela language doesn't include floating point arithmetic to the models, because the purpose of the language is to encourage abstraction from the computational aspects and focusing on the verification of process interaction [2]. So we need a more than integer-based arithmetic and it cannot be done in most of the cases.

Next, we can move to tools that use rather complex automaton models, especially timed automata. One great representative is Uppaal [4]. It offers construction of such extended automata, check invariants and can verify properties expressed with modalities (i.e. always predicate). For example, if we would like to test $z < m$ during a particular system run, we can check it dynamically by putting $z < m$ as an invariant in desired states or statically verify that goal by using a query with $E[](z < m)$. To describe the system in the Uppaal, we should implicitly create the behaviour automaton. We can introduce control variables and during transitions we can update them by calling our functions that are being written in a code which almost looks like C. The creators of Uppaal made a big step from ordinary discrete automaton – they introduce a SMC(Statistical Model Checking) extension [6] that offers making controlled non-determined transitions, adds double datatype to user's code, adds floating-point clocks type (user can specify the delta step for it in the settings) and they even target to model and verify hybrid systems by introducing time based derivatives in the invariants.

The main disadvantage of writing code for hybrid systems in Uppaal (as in some other tools) is that we should program it almost implicitly using the offered language and it is hard to write complicated ODEs or other types of mathematical models. Uppaal supports time-based derivatives, we can use for checking invariants when staying in a state (as an additional way to check the correctness of a mathematical model implementation, see Figure 6).

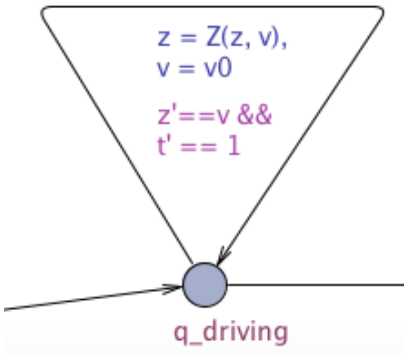


Fig. 6. Derivatives of an invariant on a Uppaal model

Lastly, we refer to the rich world of verification tools for C. Note, that a huge amount of mathematical libraries has been written in C. The modular platform for static analysis Frama-C [10] can prove a lot of types of C programs, it uses the deductive approach and extends the Hoare logic to work with pointers, memory and various type conversions. The floating point arithmetic is supported. They use a Weakest Precondition (WP) method and the verification of the program in this case will consist of calculating the weakest precondition from the end to the beginning of the function code and setting up the problem of proving the reverse derivation to the theorem prover (an internal and some externals interactive provers can be used). So, it is a very strict method and to prove the function, all the precondition, post-conditions, changes the variables, loops, internal function calls must be annotated in a special form (see the invariant with $z < m$ on Figure 7). The ISO-standardized language ACSL [1] is used.

To verify the hybrid system with Frama-C, the mathematical model for it should be solved (by direct or numerical methods) and annotated in C (and annotations can occupy huge places in a code). There is no explicit way to write it in the terms of mathematics.

A similar approach is pursuited by Ariadne [11], a framework implemented in C++. The user can encode a CPS in form of a hybrid automaton including its requirements as instances of C++ classes. More precisely, there are special classes for declaring states, transitions and invariants of the modeled system. After defining the system, the user can execute code to do reachability analysis and prove properties of the described system, especially for safety verification. Ariadne allows parametric verification, which exhaustively checks all possible values of the parameters and determines for which values the component obeys the guarantees. For setting up physical formulas in a model, the user has to use overloaded operations, which are not fully supported by the framework yet. Also, a graphical system representation is not supported yet and enforces the user to work with plain C++-code all time.

```
/*@ requires z0_0 ≥ 0;
    requires z0_0 < m;
    requires eps > 0;
    requires v0_1 > 0;
    requires dt > 0;
    ensures \result < \old(m);
 */
double model(double z0_0, double v0_1, double m, double eps, double dt)
{
  double SB = (double)0;
  double z_0 = z0_0;
  double v = v0_1;
  states q = q_start;
  double t_0 = (double)0;
  int weRun = 1;
  /*@ loop invariant z_0 ≥ 0;
      loop invariant z_0 < m;
      loop invariant t_0 ≥ 0;
      loop invariant weRun ≡ 1 ∨ weRun ≡ 0;
      loop assigns z_0, v, t_0, SB, weRun;
  */
  while (weRun) {
    SB = m - eps * v0_1;
    if (z_0 < SB) {
      t_0 = (double)0;
      /*@ loop invariant z_0 ≥ 0;
          loop invariant z_0 < m;
          loop invariant t_0 ≥ 0;
          loop invariant t_0 ≤ eps;
          loop assigns z_0, v, t_0;
      */
      while (t_0 <= eps) {
        v = v0_1;
      }
      z_0 = Z(t_0,(int)v,z_0);
      t_0 += dt;
    }
```

Fig. 7. Annotations in the simple hybrid model in C

5. Conclusion and Future Work

In this paper, we discussed one of the biggest barrier of verification tools such as KeYmaera to get widely acceptance in industry: They assume the user to be highly trained in mathematical logic and to know in detail the system's proof rules. In addition, a particular problem of KeYmaera is the representation of a proof. The actual proof of a system property can be saved by KeYmaera, but inspection of it by the user is very hard, since the key ideas of a proof are cluttered by many other proof rule applications, which are necessary to get a formal proof right.

Based on a simple but typical example, we illustrated our new approach to let the user annotate key proof facts to the system description itself. As a result, there are much more proof obligations to be proven by KeYmaera, but they are much simpler now and require much less user interaction while the formality of the proof is preserved.

So far, we treated the illustrated example as a pen-and-pencil case study. The next step will be to build a prototypical front-end tool, that allows the user to specify the system graphical as shown in Fig. 3 and which will generate the proof obligations for KeYmaera automatically.

References

[1] Baudin P. et al., *ACSL: ANSI/ISO C Specification Language. Version 1.12*, https://frama-c.com/download/acsl_1.12.pdf.

[2] *Spin: Promela reference. float - floating point numbers*, http://spinroot.com/spin/Man/float.html.

[3] Alur R. et al., "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems", *Hybrid Systems*, Lecture Notes in Computer Science, **736**, 1993, 209–229, https://doi.org/10.1007/3-540-57318-6_30.

[4] Behrmann G. et al., "A tutorial on uppaal", *Formal methods for the design of real-time systems*, Springer, 2004, 200–236.

[5] Booch G. et al., *The unified modeling language user guide – covers UML 2.0*, Second Edition, Addison Wesley object technology series, Addison-Wesley, 2005.

[6] David A. et al., "Uppaal SMC tutorial", *International Journal on Software Tools for Technology Transfer*, **17**:4 (2015), 397–415.

[7] Floyd R.W., "Assigning Meanings to Programs", *Proceedings of Symposium on Applied Mathematics*, **19** (1967), 19–32.

[8] *Hybrid Systems*, Lecture Notes in Computer Science, **736**, ed. Robert L. Grossman et al., 1993.

[9] Henzinger Thomas A., "The Theory of Hybrid Automata", *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, 1996, 278–292.

[10] Kirchner F. et al., "Frama-C: A software analysis perspective", *Formal Aspects of Computing*, **27**:3 (2015), 573–609.

[11] Nuzzo P. et al., "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems", *Proceedings of the IEEE*, **103**:11 (2015), 2104–2132.

[12] Platzer A., *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*, Springer, Heidelberg, 2010.

[13] Platzer A., "Logic and Compositional Verification of Hybrid Systems (Invited Tutorial)", *Computer Aided Verification, 23rd International Conference*, Lecture Notes in Computer Science, **6806**, 2011, 28–43.

[14] Quesel J.-D. et al., "How to Model and Prove Hybrid Systemswith KeYmaera:A Tutorial on Safety", *STTT*, **18**:1 (2016), 67-91.

[15] Rümmer Ph., *Princess Homepage*, http://www.philipp.ruemmer.org/princess.shtml.

A Generated Proof Obligations

For the running example of this paper introduced in Sect. 2. and for the additional proof contracts formulated in Sect. 3. the following proof obligations were generated (in KeYmaera syntax). For the numbering of each proof obligation (PO) the reader is referred to Sect. 3.1.

1.1. PO 1

```
\problem {
\[ R v0, ep, m, SB, v, z, t ; \] (
z < m & ep > 0 & v0 > 0
->
z < m
)
}
```

1.2. PO 2

```
\problem {
\[ R v0, ep, m, SB, v, z, t ; \] (
z < m
->
z < m
)
}
```

1.3. PO 3

```
\problem {
\[ R v0, ep, m, SB, v, z, t ; \] (
ep > 0 & v0 > 0
-> (
z < m
->
( z < m)
)
)
}
```

1.4. PO 4

```
\problem {
\[ R v0, ep, m, SB, v, z, t ; \] (
ep > 0 & v0 > 0
-> (
z < m
->
( z < m)
)
)
}
```

1.5. PO 5

```
\problem {
\[ R v0, ep, m, SB, v, z, t ; \] (
ep > 0 & v0 > 0
-> (
z < m
-> \[
SB := m - ep * v0; ? z < SB; t := 0; v := v0
\]
( z < m - ep * v0 & t = 0 & v = v0 & z < m)
)
)
}
```

1.6. PO 6

```
\problem {
\[ R v0, ep, m, SB, v, z, t ; \] (
ep > 0 & v0 > 0
-> (
z < m
-> \[
SB := m - ep * v0; ? ! z < SB; v := 0
\]
( v = 0 & z < m)
)
)
}
```

1.7. PO 7

For this proof obligation on node *driving* the user provided the additional formula
$z_{out} = z + v * tdiff \wedge t_{out} = t + tdiff$

```
\problem {
\[ R v0, ep, m, SB, v, z, t, z_out, t_out, tdiff ; \] (
ep > 0 & v0 > 0
-> (
(z < m - ep * v0 & v = v0 & t = 0 & z < m) &
(0 <= tdiff) &
(t <= ep) &
(t_out <= ep) &
(z_out = z + v * tdiff & t_out = t + tdiff)
->
z_out < m
)
)
}
```

1.8. PO 8

For this proof obligation on node *stopped* the user provided the additional formula
$z_{out} = z + v * tdiff$

```
\problem {
\[ R v0, ep, m, SB, v, z, t, z_out, t_out, tdiff ; \] (
ep > 0 & v0 > 0
-> (
(v = 0 & z < m) &
(0 <= tdiff) &
(true) &
(true) &
(z_out = z + v * tdiff)
->
z_out < m
)
)
}
```

1.9. PO 9

```
\problem {
\[ R v0, ep, m, SB, v, z, t ; \] (
ep > 0 & v0 > 0
-> (
z < m
->
( z < m)
)
)
}
```

1.10. PO 10

```
\problem {
\[ R v0, ep, m, SB, v, z, t ; \] (
ep > 0 & v0 > 0
-> (
z < m
->
( z < m)
)
)
}
```

Баар Т., Старолетов С.М., " Упрощение процесса верификации кибер-физических систем с использованием подхода с графом потока управления в средстве KeYmaera", *Моделирование и анализ информационных систем*, **25**:5 (2018), 465–480.

DOI: 10.18255/1818-1015-2018-5-465-480

Аннотация. KeYmaera является средством интерактивного доказательства теорем и используется для проверки свойств безопасности кибер-физических систем (CPS). Проверка таких свойств в интерактивном режиме может быть осложнена, поскольку доказательство осуществляется с использованием классического секвентного логического исчисления и успешное доказательство требует от пользователя глубоких знаний о доступных правилах, имеющихся в логике исчисления. Еще одним препятствием для широкого применения KeYmaera является представление текущих целей только в виде текста, что предполагает хорошую подготовку пользователя для построения успешных доказательств. В этой статье мы представляем альтернативный метод верификации для KeYmaera, который значительно повышает удобство использования и минимизирует работу пользователей. Основная идея заключается в том, чтобы позволить пользователю добавлять аннотации в виде инвариантов и контрактов к состояниям гибридной программы. В нашем подходе пользователь может использовать графический язык представления моделируемой системы, что позволяет ему не работать с чисто текстовым форматом гибридных программ, являющимся входным для средства KeYmaera. Исходя из предоставленных пользователем контрактов, можно получать доказательства, которые гораздо проще, чем исходная цель доказательств в KeYmaera, и которые могут быть обработаны в большинстве случаев полностью автоматически. Статья публикуется в авторской редакции.

Ключевые слова: кибер-физические системы, KeYmaera, контракты, верификация, гибридные системы, интерактивные доказатели теорем

Об авторах:
Баар Томас, orcid.org/0000-0002-8443-1558, профессор,
Университет прикладных технических и экономических наук г. Берлина, Германия
Wilhelminenhofstrasse 75 A, D-12459, Berlin, Germany, e-mail: thomas.baar@htw-berlin.de

Старолетов Сергей Михайлович, orcid.org/0000-0001-5183-9736, канд. физ.-мат. наук, доцент
Алтайский государственный технический университет им. И.И. Ползунова
проспект Ленина, 46, г. Барнаул, Алтайский край, 656038, Российская Федерация,
e-mail: serg_soft@mail.ru

Моделирование и анализ информационных систем. Т. 25, № 5 (2018), с. 481–490
Modeling and Analysis of Information Systems. Vol. 25, No 5 (2018), pp. 481–490

DOI: 10.18255/1818-1015-2018-5-481-490

УДК 517.9

О методах верификации и разработки программ развития сельскохозяйственных территорий

Вега Висе Х. Л., Михайлов В. Ю.

получена 20 июля 2018

Аннотация. В настоящее время повсеместными стали методы программно-целевого управления развитием различных социально-экономических систем сложной структуры, например, таких как территории сельскохозяйственного назначения. Поэтому актуальными задачами являются верификация уже созданных программ развития и разработка «правильных» программ развития таких систем, по аналогии с верификацией и разработкой правильных компьютерных программ – развитыми дисциплинами в теоретическом программировании. В данной работе для решения задачи верификации программ развития сельскохозяйственных территорий сначала строится структурная схема программы, по которой создается аксиоматическая теория, использующая аппарат алгоритмических логик Хоара. Основной проблемой при построении аксиоматической теории является разработка аксиом теории, отражающих предусловия и эффекты выполнения содержательных действий, указанных в тексте программы развития. Верификация программы развития соответствует проверке доказуемости некоторой тройки Хоара, соответствующей начальным и целевым условиям программы. Для задачи разработки правильных программ развития описывается механизм построения модели предметной области с использованием языков описания моделей семейства PDDL. Описание конкретной модели имеет чисто декларативный характер и представляет собой набор описаний предикатов и действий выбранной предметной области. Показывается, как на описанной модели с помощью интеллектуальных планировщиков, включая темпоральные планировщики типа OPTIC, автоматически строить решения целевых задач программ развития. На основе экспертных знаний и отраслевых стандартов построена модель сельскохозяйственной территории, краткое описание которой приводится в работе. Проведенные эксперименты показали эффективность предлагаемого подхода к разработке правильных программ развития.

Ключевые слова: программы развития, верификация программ развития, разработка правильных программ развития, логики Хоара, язык PDDL, интеллектуальные планировщики, модель сельскохозяйственной территории

Для цитирования: Вега Висе Х. Л., Михайлов В. Ю., "О методах верификации и разработки программ развития сельскохозяйственных территорий", *Моделирование и анализ информационных систем*, **25**:5 (2018), 481–490.

Об авторах:
Вега Висе Хорхе Луис, orcid.org/0000-0003-4323-722X, аспирант,
Казанский (Приволжский) федеральный университет,
ул. Кремлёвская, 18, г. Казань, 420008 Россия, e-mail: jvegavice@gmail.com

Михайлов Валерий Юрьевич, orcid.org/0000-0002-2778-1797, канд. физ.-мат. наук, доцент,
Казанский (Приволжский) федеральный университет,
ул. Кремлёвская, 18, г. Казань, 420008 Россия, e-mail: Valery.Mikhailov@kpfu.ru

Введение

Среди многочисленных значений слова «программа» в толковых словарях русского языка выделяются три: (1) краткое содержание работы или деятельности (например, программа передач радио или телевидения, программа концерта и т.п.); (2) план деятельности, совокупность действий и мероприятий для осуществления поставленных целей, формулировка которых также часто является частью программы (например, программа партии, программа «500 дней», программа учебной дисциплины); (3) программа для компьютера, которая представляет собой запись на некотором формальном языке алгоритма решения определенного множества задач.

Программы развития (ПР) территорий, естественно, не являются программами для компьютеров, но разрабатываются столь подробно, что значение слова «программа» в них содержит явные признаки как 2-го, так и 3-го значений из вышеуказанного списка. Поэтому представляется естественным применять к их верификации и разработке соответствующие методы, развитые в теоретическом программировании для компьютерных программ.

В данной работе предлагаются подход к применению проверки доказуемости теорем в одной из *алгоритмических логик Хоара* [1] для верификации ПР территорий, а также описание модели предметной области на языке PDDL (Planning Domain Definition Language) [2] и использование программ *темпоральных планировщиков* [3] для получения последовательности действий, позволяющей достичь желаемого состояния территории, на основе которой легко строится необходимая правильная ПР.

1. Верификация ПР

Любая ПР рассматривает территорию как объект управления, который необходимо перевести из данного начального состояния S_0 в некоторое целевое состояние S_k за k периодов времени (месяцев, кварталов, лет). Состояние территории моделируется конечным списком параметров $P=<p_1, p_2, \ldots, p_n>$, например, таких как «содержание фосфора в почве», «содержание азота в почве», «влажность почвы», «число обрабатывающих механизмов», «объем имеющихся финансовых ресурсов на развитие» и т.п. Выделяются состояния территории, относящиеся к концу определенного временного периода. Состояние территории S_t в конце периода t в ПР описывается набором действительных чисел $P_t =< p_{t1}, p_{t2}, \ldots, p_{tn} >$, где $p_{t1}, p_{t2}, \ldots, p_{tn}$ – конкретные значения параметров $p_1, p_2, \ldots, p_n$ соответственно.

В каждой ПР указывается некоторое множество базовых действий $E=\{e_1, \ldots, e_s\}$, часто называемые мероприятиями, выполнение которых в каждый период времени должно приводить к изменению состояния территории. Выполнения действия e_i в разные периоды времени могут отличаться друг от друга интенсивностью, которая зависит, например, от выделенных финансовых средств на выполнение данного действия в тот или иной период времени. Оператор $e_i(a_1, a_2, \ldots, a_k)$ обозначает выполнение действия e_i с атрибутами $a_1, a_2, \ldots, a_k$. В ПР мероприятия в любой период времени могут выполняться либо последовательно, либо параллельно друг другу.

Введем понятие **выполнимой программы**.

1) Любой оператор $e_i(a_1,a_2,\ldots,a_k)$ является выполнимой программой.

2) Если π_1 и π_2 – выполнимые программы, то $(\pi_1;\pi_2)$ и $(\pi_1 \parallel \pi_2)$ также являются выполнимыми программами.

Содержательно $(\pi_1;\pi_2)$ означает последовательное выполнение программ π_1 и π_2, а $(\pi_1 \parallel \pi_2)$ – параллельное выполнение программ π_1 и π_2. Например, выполнимая программа $((e_1(a_1)\|e_2(a_2)); (e_3(a_3)\|e_4(a_4))$ соответствует параллельному выполнению операторов $e_1(a_1)$ и $e_2(a_2)$, за которым параллельно выполняются операторы $e_3(a_3)$ и $e_4(a_4)$.

Обзор многочисленных программ развития территорий показал, что структуру каждой ПР территории можно представить в виде схемы S:

$P_0 \to \pi(1) \to P_1 \to \pi(2) \to \ldots \to P_{t-1} \to \pi(t) \to P_t \to \ldots \to P_{k-1} \to \pi(k) \to P_k$,
где часть $P_{t-1} \to \pi(t) \to P_t$ соответствует структуре временного периода t, $t = 1, \ldots, k$, P_{t-1} – состояние территории в начале периода t, P_t – состояние территории в конце периода t, $\pi(k)$ – выполнимая программа.

Под спецификацией состояния P_t будем понимать его описание ϕ_t на некотором формализованном языке L.

Пусть ϕ и ψ – формулы языка L, а π – выполнимая программа.

Будем говорить, что справедливо утверждение $\{\phi\}\pi\{\psi\}$, если всякий раз, когда перед выполнением π истинна формула ϕ, после выполнения этой программы будет истинна формула ψ.

Таким образом, проверка правильности ПР территории заключается в проверке справедливости утверждений: $\{\phi_{t-1}\}\pi(t)\{\phi_t\}$, t=1,...,k. А справедливость утверждений $\{\phi_{t-1}\}\pi(t)\{\phi_t\}$ мы будем понимать как доказуемость этих утверждений в определенной теории T, соответствующей ПР и основанной на определенной алгоритмической логике.

В качестве языка спецификаций для теории T выберем формулы вида

$$p_1 \geqslant r_1 \& p_2 \geqslant r_2 \& \ldots \& p_n \geqslant r_n, \tag{1}$$

где $p_1, \ldots, p_n$ – переменные, соответствующие параметрам, а $r_1, r_2, \ldots, r_n$ – некоторые арифметические выражения.

Правильно построенными выражениями теории T являются выражения вида $\{\phi\}\pi\{\psi\}$, где формулы ϕ и ψ имеют вид (1).

Аксиомами теории T являются выражения вида

$$\{p_{i1} \geqslant x_1 \& \ldots \& p_{im} \geqslant x_m\} e(a_1, \ldots, a_k) \\ \{p_{i1} \geqslant x_1 + g_1(a_1, \ldots, a_k) \& \ldots \& p_{im} \geqslant x_m + g_m(a_1, \ldots, a_k)\}, \tag{2}$$

где $p_{i1}, \ldots, p_{im}$ – переменные, соответствующие параметрам, $x_1, \ldots, x_m$ – произвольные значения параметров $p_{i1}, \ldots, p_{im}$, $g_j(a_1, \ldots, a_k)$ – изменение параметра p_{ij} после выполнения оператора $e(a_1, \ldots, a_k)$, j=1,...,m. Заметим, что если все атрибуты оператора $e(a_1, \ldots, a_k)$ заданы, то каждое $g_j(a_1, \ldots, a_k)$ есть некоторое действительное число (положительное или отрицательное).

Аксиомы линейного порядка также являются аксиомами теории T.

Правила вывода теории T.

П1 : $\{\phi\}\pi\{\psi\}, (\psi \to \gamma) \vdash \{\phi\}\pi\{\gamma\}$
П2 : $\{\phi\}\pi\{\psi\}, (\gamma \to \phi) \vdash \{\gamma\}\pi\{\psi\}$

$$\text{П3} : \{\phi\}\pi_1\{\gamma\}, \{\gamma\}\pi_2\{\psi\} \vdash \{\phi\}(\pi_1;\pi_2)\{\psi\}$$
$$\text{П4} : \{\phi\}\pi_1\{\psi_1\}, \{\phi\}\pi_2\{\psi_2\} \vdash \{\phi\}(\pi_1 \parallel \pi_2)\{\psi_1 \&_\phi^+ \psi_2\}$$

Здесь символ «$\rightarrow$» означает импликацию, символ «$\vdash$» - логическую выводимость, символ «$\&_\phi^+$» – операцию над формулами вида (1), вычисляемую по правилу: если $p_i \geqslant a_i$ входит в ϕ , $p_i \geqslant b_i$ входит в ψ_1 , $p_i \geqslant c_i$ входит в ψ_2 , то $p_i \geqslant a_i + (b_i – a_i) + (c_i – a_i)$ входит в формулу $\psi_1 \&_\phi^+ \psi_2$.

Никакие другие подформулы в формулу $\psi_1 \&_\phi^+ \psi_2$ не входят.

Заметим, что в схеме S спецификациями состояний P_t являются формулы вида (1), в которых все выражения $r_1, r_2, \ldots, r_n$ – действительные числа, и в выполнимых программах $\pi(t)$ атрибуты всех входящих в них операторов заданы. Не трудно показать, что задача определения доказуемости троек $\{\phi_{t-1}\}\pi(t)\{\phi_t\}$, $t = 1, \ldots, k$, в теории T является эффективно разрешимой.

Отметим, что построение множества аксиом теории T является самым содержательным и трудоемким этапом построения теории T, создаваемой для проверки правильности ПР. Очевидно, что они должны отражать мнения экспертов по развитию территорий, согласовываться с данными статистики и соответствовать определенной методологии обработки ресурсов для устойчивого территориального развития [4].

2. Разработка ПР

Для разработки ПР с/х территории нами предлагается на языке PDDL создавать *модель* этой территории, состоящую из набора предикатов и действий, соответствующих определенным способам обработки различных ресурсов территориального развития, и использовать для построенной модели алгоритмы «интеллектуального планирования», т.е. алгоритмы поиска последовательностей действий, реализация которых переведет начальное состояние территории в целевое.

2.1. Язык PDDL (Planning Domain Definition Language)

Язык PDDL [5] – язык описания областей и задач планирования, появившийся в ответ на потребность в едином, универсальном для всех планировщиков языке. Основную структуру и синтаксис PDDL унаследовал от одного из своих предшественников – языка STRIPS (Stanford Research Institute Problem Solver) [6], который был создан для решения конкретной задачи управления роботом. Язык PDDL носит описательный характер, не содержит алгоритмов как таковых и предлагает стандарт представления компактных и простых моделей для спецификации областей и задач планирования.

2.2. Описание модели в PDDL

Модель в PDDL всегда представляется в двух файлах: файл домена и файл проблемы. Файл домена содержит описание предметной области в целом: описания типов существующих объектов, описания констант, описания предикатов (свойств) и

описания действий, используемых в данной области. Файл проблемы содержит конкретную постановку задачи планирования: описание конкретных взаимодействующих объектов, начальное и целевое состояния моделируемой системы. Отделение описания домена от описания проблемы позволяет использовать модель одного и того же домена многими пользователями в решении своих задач разработки планов развития.

Файл домена содержит все возможные предикаты и действия, которые нужны для описания системы. Предикаты описываются в стандартной называющей форме, где после названия предиката перечисляются его аргументы с указанием их типов. После описания предикатов в файле домена идёт блок описания действий. Описание каждого из действий состоит из трёх основных частей: параметры действия, предусловие выполнения действия и эффект выполнения действия. Параметры содержат локальные имена переменных с указанием их типов; предусловие содержит выражение из предикатов и логических связок и должно быть истинным, чтобы действие было выполнимо; эффект содержит предикаты, соединённые связкой «и», которые станут истинными после выполнения действия.

Необходимо отметить, что PDDL постоянно эволюционирует, расширяя свои выразительные средства описания моделей. В настоящее время существует довольно большое количество планировщиков, которые способны интерпретировать модели, определенные в PDDL. Среди них можно выделить SGPlan, TFD, LPD, VHPOP, HSP, CTP, POPF, OPTIC, COLIN, FastDownward [7,8,9]. Для наших целей самым удобным и производительным оказался темпоральный планировщик OPTIC [10], разработанный для расширения языка PDDL2.1 [11], на котором мы и производили эксперименты с моделями.

Приведем формат описания простого и темпорального действий в PDDL2.1 соответственно.

```
(:action <имя действия>
 :parameters ( <список параметров действия> )
 :precondition (and ( <условие> ) ) /* условие, которое должно выполняться
                                       при запуске действия
 :effect (and ( <предикат> ) ... )
) /* конец действия
(:durative-action <имя действия>
 :parameters ( <список параметров действия> )
 :duration ( <продолжительность действия> )
 :condition (and (at start ( <условие> ) ) /* условие, которое должно выполняться
                                              при запуске действия
                 (over all ( <условие> ) ) /* условие, которое должно выполняться
                                              во время исполнения действия
                 (at end ( <условие> ) ) /* условие, которое должно выполняться
                                            в конце исполнения действия
            ) /* конец условий
 :effect (and (at start ( <предикат> ) ) ...
              (at end ( <предикат> ) )
         ) /* конец эффектов
) /* конец действия
```

Далее, в качестве простого примера приведем фрагмент экспериментальной модели работы сельскохозяйственной фермы.

```
(:action to-applyFertilizer
 :parameters (?f - fertilizer ?p - plot)
 :precondition (and (it-has THE-FARM ?f) )
 :effect (and
              ( when (it-has ?f nutrient-P) (it-has ?p nutrient-P) )
              ( when (it-has ?f nutrient-N) (it-has ?p nutrient-N) )
              ( when (it-has ?f nutrient-K) (it-has ?p nutrient-K) )
              (not (it-has THE-FARM ?f))
         )
) /* конец действия
(:durative-action to-sowPlot
 :parameters (?p - plot ?c - crop ?s - seed)
 :duration (= ?duration (time-to-sow-crop ?c) )
 :condition (and
                   (at start (plot-isUnsown ?p) )
                   (at start (it-has THE-FARM ?s) )
                   (at start (it-for ?s ?c) )
                   (at start (it-has ?p plHumidity) )
                   (at start (seed-forCrop ?s ?c) )
                   (at start (it-has ?p nutrient-P) )
                   (at start (it-has ?p nutrient-N) )
                   (at start (it-has ?p nutrient-K) )
                   (over all (not (it-has ?p plSalinity) ) )
                   (over all (not (it-has ?p plHeavyMetals) ) )
                   (over all (not (it-has ?p plOil) ) )
              )
 :effect (and
              (at end (plot-isSown ?p) )
              (at end (not (plot-isUnsown ?p) ) )
              (at end (plot-isSownWithCrop ?p ?c ?s) )
              (at end (not (it-has THE-FARM ?s)) )
              (at end (increase (total-elapsed-time) ?duration))
         )
) /* конец действия
```

В действии «to-applyFertilizer» используется предикат «(it-has ?x ?y)», который не ограничивает тип его параметров и позволяет утверждать, что «?x имеет ?y». В предусловии этого действия выражение «(and (it-has THE-FARM ?f))» означает, что действие может быть выполнено, только если на объекте «THE-FARM» (ферма) имеется удобрение «?f». В эффектах действия выражение «(when (it-has ?f nutrient-P) (it-has ?p nutrient-P))» означает, что если удобрение «?f» содержит питательное вещество «nutrient-P» (фосфор), то участок «?p» будет содержать это питательное вещество (то же самое с питательными веществами «nutrient-N» (азот) и «nutrient-K» (калий)).

Действие «to-sowPlot» является темпоральным («durative-action») и требует указания продолжительности его выполнения, используя в этом случае простую численную функцию «time-to-sow-crop ?c», которая означает, как долго длится процесс посева культуры «?c». Это действие выполняется, если в начале («at start») участок «?p» не засеян, на ферме есть семена типа «?s», участок имеет необходимую влажность, семена типа «?s» предназначены для культивирования «c» («it-for ?s ?c»), и если почва участка содержит необходимые питательные вещества «P», «N» и «K», но не содержит загрязняющие вещества «plSalinity», «plHeavyMetals» и «plOil» постоянно («over all»). В результате завершения действия («at end») участок «?p» будет засажен семенами типа «?s», а в «THE-FARM» больше не будет семян такого типа. В конце действия значение выражения «(increase (total-elapsed-time) ?duration)», равное продолжительности выполнения действия, добавляется к численному значению функции «total-elapsed-time», которая контролирует общее время выполнения целевой последовательности действий.

Отметим определенное сходство формата описания действий в PDDL и вида аксиом в теории T, строящейся для конкретной программы развития при ее верификации. С помощью описанных конструкций языка PDDL2.1 нам удалось разработать модель развития с/х территорий, кратко описанную ниже.

С точки зрения алгоритмов планирования в стандартном PDDL доступны два вида поиска решения: от начального состояния в целевое и из целевого в начальное. В первом случае, если есть несколько способов достичь цели, будет выдан кратчайший по установленной метрике (если не оговорено – по количеству действий) путь. Второй вид поиска (обратный, backward search) доступен в PDDL в силу структуры самого языка, а именно – его действий. Прямой поиск (forward search) обрабатывает действия в указанном порядке (сначала предусловие, затем применяется эффект), в то время как обратный как бы меняет эффект и предусловие местами.

2.3. Модели развития сельскохозяйственных территорий

Действующая классификация земельного фонда РФ предусматривает выделение следующих категорий пригодности земель [12]: I – земли, пригодные под пашню; II – земли, пригодные преимущественно под сенокосы; III – земли пастбищные, после улучшения могут быть пригодны под другие сельскохозяйственные угодья; IV – земли, пригодные под сельскохозяйственные угодья после коренных мелиораций; V – земли, малопригодные под сельскохозяйственные угодья; VI – земли, непригодные под сельскохозяйственные угодья; VII – нарушенные земли. Основанием для выделения категорий пригодности является качественное состояние земель и возможность их использования под основные сельскохозяйственные угодья. В отдельных случаях в зависимости от экономических и других факторов существующее использование земель может не соответствовать их намеченной пригодности. Например, земли, пригодные под пашню, если они расположены вблизи населённых пунктов или животноводческих комплексов, могут использоваться для посадки многолетних плодовых насаждений или создания долголетних культурных сенокосов и пастбищ.

Мы кратко охарактеризуем модель развития территорий первых трех типов. К разработке адекватной модели привлекались эксперты по сельскому хозяйству и

территориальному развитию и учитывались отраслевые стандарты министерства сельского хозяйства РФ ОСТ 10 294-2002 - ОСТ 10 297-2002. Ниже будем считать, что исследуемая нами с/х территория состоит из отдельных участков.

Опишем содержание *файла домена* модели «С/х территория».

Параметры и предикаты:

1) Экологические факторы участка, необходимые для нормального роста и развития растений, формирования урожая и его качества, недопущения деградации земель (закисление, засоление, переуплотнение, эрозия, дефляция, истощение запасов органического вещества и доступных для растений питательных элементов, загрязнение вредными веществами и т.д.).

2) Факторы участка, удовлетворяющие потребности сельскохозяйственных культур с учетом их биологических особенностей в питательных элементах (N, P, K, Ca, Mg, S, микроэлементы), воде, воздухе, тепле, фитосанитарных и эколого-токсикологических условий.

3) Высокопродуктивные, адаптированные к местным условиям сорта (гибриды) с/х культур.

Действия:

Группа действий 1: Осуществление агротехнических, агрохимических, мелиоративных, фитосанитарных, противоэрозионных и иных мероприятий по обеспечению плодородия почв;

Предусловия: определенные потребности с/х участка в минеральных удобрениях, химических мелиорантах, пестицидах и других агрохимикатах, органических удобрениях, оборудовании и машинах.

Эффекты: повышение плодородия почвы.

Группа действий 2: Мероприятия по реабилитации земель сельскохозяйственного назначения, загрязненных тяжелыми металлами, пестицидами, радиоактивными веществами и другими токсикантами;

Предусловия: загрязненность с/х участка, наличие соответствующего оборудовании и машин;

Эффекты: нормализация экологического состояния почвы.

Группа действий 3: Мероприятия по оптимальному размещению посевов с/х культур, формированию севооборотных массивов, выявлению малопродуктивных земель, трансформации пашни в менее интенсивные виды угодий.

Группа действий 4: Технологические действия возделывания сельскохозяйственных культур с учетом складывающихся погодных и хозяйственно-экономических условий, минерального питания растений, фитосанитарного обследования посевов, запасов продуктивной влаги и плотности почвы в период вегетации растений.

Эффекты: повышение качества и урожайности культур, сокращение затрат на их производство, предотвращение загрязнения окружающей среды средствами химизации.

Описание *файла проблемы* модели «С/х территория».

Здесь описывается набор земельных участков, составляющих территорию, и их

начальные характеристики, включающие состав почв, близость рынков, близость источников чистой воды и др.

Кроме этого, в этом файле указываются целевые показатели развития территории:

1) Улучшение качества участков.

Здесь качество участка – это комплексная характеристика участка по уровню его плодородия и производительной способности, вычисляемая на основе стандартных характеристик земли (гранулометрический состав, эрозия, засоление, избыточное увлажнение, каменистость и т.д.).

2) Планируемая экономическая выгода с/х производства на данной территории.

Наши эксперименты с построенной моделью «С/х территория» и использованием планировщика OPTIC, разработанного в King's College London, показали успешность применения описанного подхода к созданию ПР как для с/х компаний, так и для специфических территориально-экономических кластеров.

Список литературы / References

[1] Hoare C.A.R., "An Axiomatic Basis for Computer Programming", *Communications of the ACM*, **12**:10 (1969), 576–580.

[2] McDermott D.V., "PDDL—The Planning Domain Definition Language", *Tech. Rep. TR-98-003/DCS TR-1165*, Yale Center for Computational Vision and Control, 1998.

[3] Coles A.J. et al., "Temporal Planning in Domains with Linear Processes", *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009, 1671–1676.

[4] Vargas Rodriguez H. et al., "Metodología para el uso y manejo social del recurso tierra como contribución al desarrollo local sostenible", *Revista de Gestión del conocimiento y el desarrollo local – CEDAR/UNAH*, **1** (2014), 34–38; [Vargas Rodriguez H. et al., "Methodology for the use and social handling of the resource land as a contribution to the sustainable local development", *Knowledge management and local development magazine – CEDAR/UNAH*, **1** (2014), 34–38, (in Spanish).]

[5] Helmert M., *An Introduction to PDDL*, Tech. rep., https://www.cs.toronto.edu/~sheila/2542/s14/A1/introtopddl2.pdf.

[6] Fikes R., Nilsson N., "STRIPS: a new approach to the application of theorem proving to problem solving", *Artificial Intelligence*, **2** (1971), 189–208.

[7] Gerevini A., Long D., *Plan constraints and preferences in PDDL3*, Tech. Rep., Dept. of Electronics for Automation, University of Brescia, Italy, 2005.

[8] Cresswell S., Coddington A., "Compilation of LTL goal formulas into PDDL", *Proceedings of the 16-th European Conference on Artificial Intelligence*, 2004, 985–986.

[9] Coles A.J. et al., "COLIN: Planning with Continuous Linear Numeric Change", *Journal of Artificial Intelligence Research*, **44**:1 (2012), 1–96.

[10] *OPTIC: Optimising Preferences and Time-Dependent Costs*, https://nms.kcl.ac.uk/planning/software/optic.html.

[11] Fox M., Long D., "PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains", *Journal of Artificial Intelligence Research*, **20**:1 (2003), 61–124.

[12] *Категории пригодности земель*, http://geolike.ru/page/gl_3219.htm; [*Kategorii prigodnocti zemel*, http://geolike.ru/page/gl_3219.htm, (in Russian).]

Vega Vice J. L., Mikhailov V. Y., "On Methods in the Verification and Elaboration of Development Programs for Agricultural Territories", *Modeling and Analysis of Information Systems*, **25**:5 (2018), 481–490.

DOI: 10.18255/1818-1015-2018-5-481-490

Abstract. Nowadays, the methods of program-targeted management for the development of various socio-economic systems of complex structure, such as agricultural areas, have become universal. Therefore, the current tasks at hand are the verification of already created development programs and the development of "proper" programs for the development of such systems, by analogy with the verification and development of proper computer programs through developed disciplines in theoretical programming. In this paper, in order to solve the problem of verification of development programs for agricultural territories, a structural scheme of the program is first constructed, through which the axiomatic theory is created, using Hoare's algorithmic logic system. The main problem in the construction of the axiomatic theory is the development of the axioms of the theory reflecting the preconditions and effects of the implementation of meaningful actions indicated in the text of the development program. The verification of the development program corresponds to the probability of some Hoare triplet, according to the initial and target conditions of the program. For the task of elaboration of the right development programs, the mechanism for constructing a domain model using the PDDL family description languages is described. The description of a specific model is purely declarative in nature and consists of descriptions of predicates and actions of the chosen subject area. In this paper, it is shown how on the described model with the help of intelligent planners, including temporal planners such as OPTIC, solutions to the targets of development programs can be automatically built. Based on expert knowledge and activity standards, a model of an agricultural territory is constructed, a brief description of which is given in the work. The conducted experiments showed the effectiveness of the proposed approach for the development of proper development programs.

Keywords: development programs, verification of development programs, development of proper development programs, Hoare's logic, PDDL language, intelligent planners, model of agricultural territories

On the authors:
Jorge L. Vega Vice, orcid.org/0000-0003-4323-722X, graduate student,
Kazan (Volga region) Federal University,
18 Kremlyovskaya str., Kazan 420008, Russia, e-mail: jvegavice@gmail.com

Valery Y. Mikhailov, orcid.org/0000-0002-2778-1797, PhD,
Kazan (Volga region) Federal University,
18 Kremlyovskaya str., Kazan 420008, Russia, e-mail: Valery.Mikhailov@kpfu.ru

Моделирование и анализ информационных систем. Т. 25, № 5 (2018), с. 491–505
Modeling and Analysis of Information Systems. Vol. 25, No 5 (2018), pp. 491–505

DOI: 10.18255/1818-1015-2018-5-491-505

УДК 004.052.42

Автоматизация верификации С-программ с использованием символического метода элиминации инвариантов циклов

Кондратьев Д. А.[1], **Марьясов И. В.**, **Непомнящий В. А.**

получена 10 сентября 2018

Аннотация. При дедуктивной верификации программ, написанных на императивных языках программирования, особую сложность вызывает порождение и доказательство условий корректности, соответствующих циклам, поскольку каждый из них должен быть снабжён инвариантом, построение которого часто является нетривиальной задачей. Методы синтеза инвариантов циклов, как правило, носят эвристический характер, что затрудняет их применение. Альтернативой является символический метод элиминации инвариантов циклов, предложенный В.А. Непомнящим в 2005 году. Его идея состоит в представлении тела цикла в виде специальной операции замены при выполнении определённых ограничений. Такая операция в символической форме выражает действие цикла, что позволяет ввести в аксиоматическую семантику правило вывода для циклов, не использующее инварианты. В данной работе представлено дальнейшее развитие этого метода. Он расширяет метод смешанной аксиоматической семантики, предложенный для верификации C-light программ. Данное расширение включает в себя метод верификации итераций над изменяемыми массивами с возможным выходом из тела цикла в C-light программах. Метод содержит правило вывода для итерации без инвариантов циклов. Данное правило было реализовано в генераторе условий корректности, являющемся частью системы автоматизированной верификации C-light программ. Для проведения автоматического доказательства в используемой системе ACL2 были разработаны и реализованы два алгоритма: первый порождает операцию замены на языке системы ACL2, а второй генерирует вспомогательные леммы, позволяющие системе ACL2 успешно доказать получаемые условия корректности в автоматическом режиме. Применение вышеуказанных методов и алгоритмов проиллюстрировано примером.

Ключевые слова: Си-лайт, инварианты циклов, смешанная аксиоматическая семантика, финитная итерация, массивы, ACL2, спецификация, верификация, логика Хоара

Для цитирования: Кондратьев Д. А., Марьясов И. В., Непомнящий В. А., "Автоматизация верификации С-программ с использованием символического метода элиминации инвариантов циклов", *Моделирование и анализ информационных систем*, **25**:5 (2018), 491–505.

Об авторах: Кондратьев Дмитрий Александрович, orcid.org/0000-0002-9387-6735, аспирант,
Институт систем информатики им. А.П. Ершова СО РАН,
пр. Академика Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: apple-66@mail.ru

Марьясов Илья Владимирович, orcid.org/0000-0002-2497-6484, канд. физ.-мат. наук,
Институт систем информатики им. А.П. Ершова СО РАН,
пр. Академика Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: ivm@iis.nsk.su

Непомнящий Валерий Александрович, orcid.org/0000-0003-1364-5281, канд. физ.-мат. наук,
Институт систем информатики им. А.П. Ершова СО РАН,
пр. Академика Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: vnep@iis.nsk.su

Благодарности:
[1] Автор частично поддержан грантом РФФИ № 17-01-00789.

Введение

В настоящее время верификация С-программ является актуальной проблемой программирования. Некоторые проекты (например [2, 4]) предлагают различные подходы. Но ни один из них не содержит каких-либо методов автоматической верификации программ с циклами без инвариантов. Как известно, для того чтобы верифицировать программы с циклами, пользователь должен предоставить инварианты, построение которых часто является сложной задачей. Туэрк [15] предложил использовать пред- и постусловия для циклов типа **while**, однако их построение также целиком лежит на пользователе. Ли и другие [10] разработали обучающийся алгоритм генерации инвариантов циклов, тем не менее, их метод не допускает наличие оператора **break** в теле цикла, а также операций над массивами.

Мы рассматриваем циклы с определёнными ограничениями [14]. Мы расширяем наш метод смешанной аксиоматической семантики языка C-light [1] правилом для верификации таких циклов, основанным на операции замены [9, 14]. С помощью этого правила порождаются условия корректности специального вида.

В нашей статье [12] мы представили:

1. Метод верификации программ над неизменяемыми массивами с выходом из цикла.
2. Рекурсивный алгоритм построения операции замены.
3. Стратегию интерактивного доказательства условий корректности.

В процессе наших исследований появились две цели: рассмотреть случай изменяемых структур данных и разработать автоматизированные методы доказательства условий корректности. Данная статья представляет следующие результаты:

1. Метод верификации программ над изменяемыми массивами с выходом из цикла.
2. Рекурсивный алгоритм построения операции замены для таких программ.
3. Стратегия автоматизации доказательства условий корректности.

На стадии доказательства мы используем систему ACL2 [6] вместо SMT-решателей. Поэтому был разработан новый алгоритм построения операции замены, который транслирует инструкции тела цикла на языке C-light в конструкции входного языка системы ACL2. Также был разработан и реализован эвристический метод генерации вспомогательных утверждений, позволяющих системе ACL2 успешно доказывать условия корректности в автоматическом режиме.

1. Финитная итерация над изменяемыми структурами данных и операция замены

Метод элиминации инвариантов циклов для финитной итерации был предложен в [14]. Он состоит из четырёх случаев:

1. Финитная итерация над неизменяемыми структурами данных без выхода из цикла.

2. Финитная итерация над неизменяемыми структурами данных с выходом из цикла.

3. Финитная итерация над изменяемыми структурами данных без / с выходом из цикла.

4. Финитная итерация над иерархическими структурами данных с выходом из цикла.

Первый случай был рассмотрен в [11], второй — в [12]. В этой статье мы рассматриваем третий случай.

Напомним понятие структур данных, которые содержат конечное число элементов. Пусть $memb(S)$ обозначает мультимножество элементов структуры S, а $|memb(S)|$ — мощность мультимножества $memb(S)$. Для структуры S определены следующие операции:

1. $empty(S) = true$ тогда и только тогда, когда $|memb(S)| = 0$.

2. $choo(S)$ возвращает элемент из $memb(S)$, если $\neg empty(S)$.

3. $rest(S) = S'$, где S' — структура типа S и $memb(S') = memb(S) \setminus \{choo(S)\}$, если $\neg empty(S)$.

Множества, последовательности, списки, строки, массивы, файлы и деревья являются типичными примерами структур данных.

Пусть $\neg empty(S)$, тогда положим $vec(S) = [s_1, s_2, \ldots, s_n]$, где $memb(S) = \{s_1, s_2, \ldots, s_n\}$ и $s_i = choo(rest^{i-1}(S))$ для $i = 1, 2, \ldots, n$.

Рассмотрим оператор

$$\textbf{for x in S do v} := \textbf{body}(\mathbf{v}, \mathbf{x})\ \textbf{end},$$

где S — структура, x — переменная типа «элемент S», v — вектор переменных цикла, не содержащий x, а $body$ представляет вычисление, реализуемое телом цикла, которое не изменяет x и завершается для каждого $x \in memb(S)$. Структура S может быть изменена следующим образом. Тело цикла может содержать только операторы присваивания, условные операторы (в том числе вложенные) и операторы выхода из цикла **break**. Такой цикл **for** называется финитной итерацией.

Операционная семантика такого оператора выглядит следующим образом. Пусть v_0 — вектор начальных значений переменных из v. Если $empty(S)$ истинно, то результат итерации $v = v_0$. Иначе $vec(S) = [s_1, s_2, \ldots, s_n]$, тогда тело цикла последовательно итерирует таким образом, что x принимает значения $s_1, s_2, \ldots, s_n$, а $body(v, s_j)$ может изменять $s_1, s_2, \ldots, s_{j-1}$.

Для того чтобы выразить результат действия итерации, определим операцию замены $rep(v, S, body, n)$, где $rep(v, S, body, 0) = v$, $rep(v, S, body, i) = body(rep(v, S, body, i-1), s_i)$ для всех $i = 1, 2, \ldots, n$, если $\neg empty(S)$.

В [14] были доказаны теоремы, выражающие важные свойства операции замены.

Правило вывода для финитной итерации имеет вид:

$$\frac{E, SP \vdash \{P\}\,\mathbf{A};\{Q(v \leftarrow rep(v, S, body))\}}{E, SP \vdash \{P\}\,\mathbf{A};\ \mathbf{for\ x\ in\ S\ do\ v} := \mathbf{body(v, x)\ end}\{Q\}}$$

где **A** — операторы программы перед циклом. Мы используем обратное прослеживание: мы двигаемся от конца программы до её начала, удаляя самый правый оператор (на верхнем уровне), применяя соответствующее правило вывода смешанной аксиоматической семантики [1] языка C-light. E называется окружением, которое содержит информацию о текущей функции (её идентификатор, тип и тело), которую верифицируют, информацию о текущем блоке и идентификатор метки, если ранее был встречен оператор перехода **goto**. SP — спецификация программы, которая включает все предусловия, постусловия и инварианты циклов и помеченных операторов.

2. Входной язык системы ACL2

Входным языком системы ACL2 [6] является аппликативный диалект языка Common Lisp, в котором поддерживается только функциональная парадигма и отсутствует поддержка императивной. Далее под языком ACL2 будем понимать входной язык системы ACL2.

Удобным способом задания операции замены *rep* является трансляция каждой конструкции цикла в семантически эквивалентную композицию конструкций языка ACL2. Так как конструкции цикла написаны в императивной парадигме, то прямых аналогов им в языке ACL2 нет. Поэтому рассмотрим те конструкции языка ACL2, которые позволяют моделировать используемые в финитных итерациях конструкции языка C-light.

Так как язык Common Lisp ориентирован на обработку списков, то массивы языка C-light мы моделируем с помощью списков. При этом мы используем две операции для обработки индексированных последовательностей. Рассмотрим функции *nth* и *update-nth*, реализующие эти операции. Если i — индекс, l — список, то $(nth\ i\ l)$ — значение i-го элемента списка l. Если *expr* — выражение языка ACL2, то *(update-nth i expr l)* — новый список, который совпадает со списком l за исключением i-го элемента, значением которого является значение *expr*.

Рассмотрим используемые в языке ACL2 операции над списками, позволяющие моделировать соответствующие операции над массивами. Предикат *integer-listp* истинен тогда и только тогда, когда аргументом является список и все его элементы имеют целочисленный тип. Предикат *equal* истинен тогда и только тогда, когда равны и длины списков, и их элементы. Функция *length* вычисляет длину списка.

Моделировать новые типы данных позволяет библиотека *fty* языка ACL2, в которой работа с типизированными переменными сводится к использованию специальных функций. Каждому типу соответствует предикат, определяющий принадлежность к данному типу, функция приведения к данному типу и отношение эквивалентности. Такие типы можно задать с помощью специальных конструкций библиотеки *fty*. Макрос *fty::defprod* задаёт тип, аналогичный инструкции **struct** языка C-light. Если *st* — такой тип, то будут сгенерированы связанные с типом *st* функции с названиями *st-p*, *st-fix* и *st-equiv* соответственно. Также с помощью *fty::defprod* будет сгенерирован конструктор, макросы *make-st, change-st* и функции

доступа к значениям полей. Пусть fd — поле структуры s, имеющей тип st. Тогда *(change-st s :fd expr)* — новая структура типа st, совпадающая со структурой s за исключением поля fd, значением которого является значение $expr$. Если fd — единственное поле структуры st, то *(make-st :fd expr)* — новая структура типа st, у которой значением поля fd является значение $expr$. Тогда *(st->fd s)* значение поля fd структуры s. Другим способом доступа к данному значению является $s.fd$.

Макрос $b*$ позволяет промоделировать последовательное исполнение инструкций. Он является расширением макроса $let*$ языка ACL2, позволяющего удобным образом задать вложенный let. Рассмотрим общий вид конструкции $let*$:

$$(let^* \ ((var_1 \ term_1) \ ... \ (var_n \ term_n)) \ body),$$

где все var_i являются переменными (не обязательно различными), $body$ и все $term_i$ являются выражениями языка ACL2. Эта конструкция эквивалентна следующей:

$$(let \ ((var_1 \ term_1)) \ldots (let \ ((var_n \ term_n)) \ body) \ ...).$$

Таким образом, связывание переменных val_i со значениями соответствующих выражений $term_i$ происходит последовательно. Значением такой конструкции является значение выражения $body$. Отметим, что каждая пара $(var_i \ term_i)$ называется связыванием переменной var_i и значения выражения $term_i$, а переменная var_i называется локальной переменной в блоке $let*$.

Рассмотрим общий вид конструкции $b*$:

(b <list-of-bindings>.<list-of-result-forms>)* .

где *<list-of-bindings>* — список конструкций *binding*, *<list-of-result-forms>* — список выражений языка ACL2. Значением конструкции $b*$ является последнее выражение списка *<list-of-result-forms>*. По аналогии с конструкцией $let*$ операции *binding* выполняются последовательно. При этом, общим видом конструкции *binding* является

(<binder-form> [<expression>]) ,

где *<binder-form>* — конструкция *b*-binder*, *<expression>* — выражение языка ACL2. Так как $b*$ является расширением $let*$, то частным случаем конструкции *binding* является связывание переменной и значения выражения. При этом переменная становится локальной переменной блока $b*$.

Таким образом, одним из возможных видов конструкции *<binder-form>* является переменная. Другим возможным видом конструкции *<binder-form>* является (*when condition*), где *condition* — логическое выражение языка ACL2. Пусть блок $b*$ содержит *binding*-конструкцию ((*when condition*) *expression*). Если выражение *condition* истинно, то следующие за данной конструкцией операции *binding* не исполняются, а значением блока $b*$ является *expression*. Такая конструкция позволяет моделировать изменение потока управления C-light программы.

Язык ACL2 поддерживает многозначные функции. Конструкция mv позволяет функции возвратить более одного значения. Для связывания многих переменных с результатом исполнения многозначных функций используется следующая конструкция *binding*:

((mv x_1 x_2 ... x_{n-1} x_n*) (form-returning-n-values))* ,

где x_i — переменная для каждого i $(1 \leq i \leq n)$, *(form-returning-n-values)* — вызов функции, возвращающий n значений. Таким образом i-я переменная x связывется с i-м значением *(form-returning-n-values)*.

Язык ACL2 обеспечивает поддержку пользовательских теорий. Формулы задаются с помощью логических связок *not* (логическое «не»), *or* (логическое «или»), *and* (логическое «и»), *implies* (импликация). В таких логических выражениях удобно использовать предикаты для проверки принадлежности переменной определённому типу. Например, предикат *integerp* проверяет принадлежность аргумента целочисленному типу. Конструкция $define$ задаёт функцию на языке ACL2, а также позволяет вводить леммы о ней. С помощью конструкции $defrule$ задаются леммы, теоремы и специальные секции, содержащие информацию, которая помогает ACL2 доказать теорему. Например, в секции *prep-lemmas* можно задать леммы, позволяющие доказать определяемую конструкцией $defrule$ теорему. Секция *enable* позволяет при доказательстве использовать леммы о функциях, определённых в $define$. Секция *induct* разрешает проведение доказательства по индукции. Аргументом данной секции служит применение специальной функции к переменным теоремы. Рекурсивное определение такой функции задаёт схему индукции. Например, рекурсивное определение библиотечной функции *dec-induct* определяет классическую схему индукции по натуральному параметру с шагом 1.

3. Генерация операции замены для одномерных массивов на входном языке системы ACL2

Пусть S — одномерный массив из n элементов простого типа языка C-light и $S \in v$. Рассмотрим специальный случай финитной итерации над массивом S:

$$\mathbf{for}\,(\mathbf{i} = \mathbf{0}; \mathbf{i} < \mathbf{n}; \mathbf{i}++)\ \mathbf{v} := \mathbf{body}(\mathbf{v}, \mathbf{i})\ \mathbf{end},$$

где тело цикла является допустимой конструкцией.

Допустимая конструкция — это один из следующих операторов языка C-kernel:

1. Пустой оператор, в том числе пустой блок.
2. Оператор выхода из цикла **break**;.
3. Оператор присваивания $\mathbf{a} = \mathbf{b}$;, где a — переменная простого типа, либо имеет вид $S[i]$, b — выражение языка C-kernel.
4. Условный оператор **if** (**a**) **b**, где a — выражение языка C-kernel, b — допустимая конструкция.
5. Условный оператор **if** (**a**) **b else c**, где a — выражение языка C-kernel, b и c — допустимые конструкции.
6. Блок $\{\mathbf{a_1}\ \mathbf{a_2}\ ...\ \mathbf{a_{k-1}}\ \mathbf{a_k}\}$, где a_r — допустимая конструкция для каждого r: $1 \leq r \leq k$.

В вектор v входит массив S и переменные простого типа, которые могут изменяться в теле цикла. Введём функцию w, её аргументом является допустимая конструкция, и она возвращает множество элементов вектора v для данной конструкции. Определим такую функцию для каждого вида допустимой конструкции *op*:

1. Если *op* — это пустой оператор (в том числе пустой блок), то $w(op) = \emptyset$.
2. Если *op* — это **break**;, то $w(op) = \emptyset$.
3. Если *op* – это **a = b**;, где a — переменная простого типа либо имеет вид $S[i]$, b — выражение языка C-kernel, то возможны два варианта:
   (a) Если a — переменная простого типа, то $w(op) = \{a\}$.
   (b) Если a имеет вид $S[i]$, то $w(op) = \{S\}$.
4. Если *op* — это **if (a) b**, то $w(op) = w(b)$.
5. Если *op* — это **if (a) b else c**, то $w(op) = w(b) \cup w(c)$.
6. Если *op* — это $\{\mathbf{a_1\ a_2\ \ldots\ a_{k-1}\ a_k}\}$, то $w(op) = w(a_1) \cup w(a_2) \cup \ldots \cup w(a_{k-1}) \cup w(a_k)$.

Пусть T — результат применения функции w к телу цикла. Рассмотрим произвольное упорядочивание элементов множества T. Обозначим его как вектор v.

Генерация функции *rep* состоит из трёх этапов. На первом шаге осуществляется генерация типа данных $frame$ с помощью конструкции *fty::defprod*. Этот тип данных представляет собой структуру, полями которой являются элементы вектора v, а также булевское поле **loop-break**. Также конструкция *fty::defprod* генерирует макросы *make-frame* и *change-frame*, функции *frame-p*, *frame-fix*, *frame-equiv* и функции доступа к значениям полей.

Значения полей переменной типа $frame$ являются значениями соответствующих переменных в ходе исполнения цикла. Поле **loop-break** истинно тогда и только тогда, когда сработал оператор **break**. Таким образом, переменная типа $frame$ хранит значения вектора v в ходе исполнения цикла. Поэтому необходим способ задания начальных значений элементов вектора v, то есть значений перед исполнением цикла. Для этого на втором этапе с помощью конструкции *make-frame* происходит генерация функции *frame-init*. Эта функция возвращает структуру типа $frame$ с заданными начальными значениями полей. Значением поля **loop-break** у такой структуры является nil. Это означает, что перед исполнением цикла оператор **break** не срабатывает.

На третьем этапе происходит генерация функции *rep* по телу цикла следующим образом. Первым аргументом функции *rep* является номер итерации. При использовании функции *rep* в условии корректности в качестве значения первого аргумента используется n, так как n — номер последней итерации. Вторым аргументом функции *rep* является переменная типа $frame$, поля которой соответствуют начальным значениям элементов вектора v. Таким образом, при использовании функции *rep* в условии корректности значением второго аргумента является *frame-init*(v_0), где v_0 — вектор начальных значений элементов v.

Пусть значением первого аргумента функции rep является m. Тогда функция rep должна возвращать такую структуру типа $frame$, что значения ее полей будут значениям вектора v после m итераций цикла.

Будем считать, что нулевая итерация цикла соответствует значениям элементов вектора v перед исполнением цикла. Поэтому

$$rep(0,\ frame\text{-}init\ (v_0)) = frame\text{-}init\ (v_0)\ .$$

Это ограничивает глубину рекурсии.

В случае выхода из цикла на итерации l $(0 < l \leq n)$ при исполнении оператора **break** будем считать, что итерации цикла продолжаются, но значения v не изменяются на таких итерациях j, что $l \leq j \leq n$. Зададим rep так, что

$$rep(l,\ frame\text{-}init\ (v_0)) = rep(j,\ frame\text{-}init\ (v_0))\ .$$

Поэтому необходимо проверять на истинность значение поля **loop-break** у возвращённой на предыдущей итерации структуры типа $frame$. В случае, когда выход из цикла происходит на текущей итерации, необходимо прекратить исполнение и вернуть такую структуру типа $frame$, что её поле **loop-break** будет истинным.

Каждую переменную функции rep и ее аргумент типа $frame$ обозначим через fr. При необходимости изменить значения полей переменной fr создаётся новая переменная fr с новыми значениями полей. Это позволяет удобным образом обрабатывать элементы вектора v.

Алгоритм генерации функции rep состоит из двух шагов. На первом шаге используется функция $generate\_rep\_one$, которая строит сигнатуру функции rep и конструкцию $b*$. Эта конструкция $b*$ является самым верхним по уровню вложенности блоком кода функции rep на языке ACL2. Этот блок соответствует составному оператору, являющемуся телом цикла. Такая конструкция позволяет промоделировать последовательное исполнение следующих инструкций: проверку условия выхода из рекурсии с помощью конструкции $when$, рекурсивный вызов rep для предыдущей итерации, результат которого сохраняется в переменной fr, проверку с помощью конструкции $when$ выхода из цикла на предыдущей итерации.

Каждая такая инструкция блока $b*$ является конструкцией *binding*. Следующие за ними в этом блоке $b*$ инструкции *binding* моделируют последовательное исполнение тела цикла, совершая операции над структурой fr. Они будут сгенерированы функцией $generate\_rep$ на втором шаге алгоритма. Кроме того, на первом шаге алгоритма функция $generate\_rep\_one$ генерирует выражение fr, являющееся значением данного блока $b*$. Таким образом, если не было выхода из цикла, то значением функции rep на текущей итерации будет переменная fr, полученная при применении к результату предыдущей итерации конструкций, соответствующих телу цикла.

На втором шаге алгоритма используется функция $generate\_rep$, генерирующая конструкции, соответствующие телу цикла. Отметим, что для любой финитной итерации данного вида на первом шаге алгоритма генерируется один и тот же код на языке ACL2. То есть различия между разными финитными итерациями учитываются не на первом, а на втором шаге алгоритма.

Функция $generate\_rep$ осуществляет трансляцию тела цикла на язык ACL2. Она принимает в качестве аргумента допустимую конструкцию и транслирует её на

язык ACL2. Далее под объемлющим блоком для конструкции *binding* будем понимать блок $b*$, в котором она находится. Функция $c\_kernel\_translator$ осуществляет трансляцию C-kernel выражения на язык ACL2. Определим функцию $generate\_rep$ для каждого вида допустимой конструкции op:

1. Если op – это пустой оператор (в том числе пустой блок), то результатом $generate\_rep(op)$ будет $(fr\ fr)$. Данная запись является конструкцией *binding* в объемлющем блоке $b*$. Она означает, что внутри блока создается новая переменная fr со значением прежней и с новой областью видимости. То есть пустой оператор (в том числе пустой блок) не меняет значения элементов вектора v.

2. Если op — это **break**;, то результатом $generate\_rep(op)$ будет

$$\textit{(fr (change-frame fr :loop-break t)) ((when t) fr)}\ .$$

   Данная запись представляет собой две последовательно идущие конструкции *binding* в объемлющем блоке $b*$. Первая конструкция *binding* означает, что внутри блока создается новая переменная fr со значением *(change-frame fr :loop-break t)* и с новой областью видимости. То есть у новой переменной fr поле *loop-break* стало истинным.

   Вторая конструкция *binding* означает, что при соблюдении условия *when* происходит выход из объемлющего блока $b*$. Так как таким условием *when* является t, то всегда происходит выход из объемлющего блока $b*$ при срабатывании данной второй конструкции. То есть при срабатывании оператора **break** происходит выход из цикла. Возвращаемым значением такого объемлющего блока $b*$ будет являться переменная fr, поле *loop-break* которой будет указывать на то, что необходимо также произвести выход из блоков, в которые вложен данный объемлющий блок (если они существуют).

3. Если op — это $\mathbf{a}\ =\ \mathbf{b}$;, где a — переменная простого типа либо имеет вид $S[i]$, b — выражение языка C-kernel, то возможны два варианта:

   (a) Если a — переменная простого типа, то результатом $generate\_rep(op)$ будет

$$\textit{(fr (change-frame fr :a c\_kernel\_translator(b)))}\ .$$

   Данная запись является конструкцией *binding* в объемлющем блоке $b*$. Она означает, что внутри блока создаётся новая переменная fr со значением *(change-frame fr :a c_kernel_translator(b))* и с новой областью видимости. То есть у новой переменной fr поле a принимает значение выражения b. Таким образом, оператор присваивания меняет значение переменной a вектора v.

   (b) Если a имеет вид $S[i]$, то результатом $generate\_rep(op)$ будет

$$\textit{(fr (change-frame fr :S (update-nth i c\_kernel\_translator(b)}\\ \textit{(frame->S fr))))}\ .$$

Данная запись является конструкцией *binding* в объемлющем блоке $b*$. Она означает, что внутри блока создается новая переменная fr со значением

(change-frame fr :S (update-nth i c_kernel_translator(b) (frame->S fr)))

и с новой областью видимости. То есть у новой переменной fr поле S принимает значение последовательности S, в которой на месте элемента с индексом i стоит значение выражения b. Таким образом оператор присваивания изменяет массив S из вектора v.

4. Если *op* — это **if** (**a**) **b**, то результатом *generate_rep*(*op*) будет

(fr (if c_kernel_translator(a) (b (generate_rep(b)) fr) fr)) ((when (frame->loop-break fr)) fr) .*

Данная запись представляет собой две последовательно идущие конструкции *binding* в объемлющем блоке $b*$. Первая конструкция *binding* означает, что внутри блока создается новая переменная fr с новой областью видимости и со значением

(if c_kernel_translator(a) (b(generate_rep(b)) fr) fr) .*

То есть при выполнении условия a значением переменной fr станет

(b (generate_rep(b)) fr) ,*

где $generate\_rep(b)$ — результат трансляции b на входной язык системы ACL2. Отметим, что любой оператор b (в том числе составной) транслируется в одну или две *binding* конструкции. Блок $b*$ в ACL2-коде для оператора if является объемлющим для этих *binding* конструкций. Таким образом, при выполнении условия a значение вектора v будет определяться положительной ветвью условного оператора.

При невыполнении условия a значением переменной fr станет предыдущее значение fr. Таким образом, при невыполнении условия a значение вектора v не изменится.

Вторая конструкция *binding* означает, что при соблюдении условия $when$ происходит выход из объемлющего блока $b*$. Так как таким условием $when$ является *(frame->loop-break fr)*, то выход из объемлющего блока $b*$ происходит при истинности поля *loop-break* переменной fr. То есть если при исполнении оператора сработала инструкция $break$, то происходит выход из объемлющего блока $b*$. Возвращаемым значением такого объемлющего блока $b*$ будет являться переменная fr, поле $loop - break$ которой будет указывать на то, что необходимо также произвести выход из блоков, в которые вложен данный объемлющий блок (если они существуют).

5. Если *op* — это **if** (**a**) **b else c**, то результатом *generate_rep*(*op*) будет

$$\begin{array}{c}\textit{(fr (if c\_kernel\_translator(a) (b*(generate\_rep(b)) fr)}\\ \textit{(b*(generate\_rep(c)) fr))) ((when (frame->loop-break fr)) fr)}\ .\end{array}$$

Данная запись представляет собой две последовательно идущие конструкции *binding* в объемлющем блоке $b*$. Первая конструкция *binding* определяется по аналогии с пунктом 4.

Определение второй конструкции *binding* совпадает с определением из пункта 4.

6. Если *op* — это $\{\mathbf{a_1}\ \mathbf{a_2}\ \ldots\ \mathbf{a_{k-1}}\ \mathbf{a_k}\}$, то результатом *generate_rep*(*op*) будет

$$\begin{array}{c}(fr\ (b^*(generate\_rep(a_1)\ generate\_rep(a_2)\ldots generate\_rep(a_{k-1})\\ generate\_rep(a_k))fr))((when\ (frame\text{->}loop\text{-}break\ \ fr))\ \ fr)\ .\end{array}$$

Данная запись представляет собой две последовательно идущие конструкции *binding* в объемлющем блоке $b*$. Первая конструкция *binding* означает, что внутри блока создается новая переменная fr с новой областью видимости и со значением

$$\begin{array}{c}(b^*(generate\_rep(a_1)\ generate\_rep(a_2)\ldots\\ generate\_rep(a_{k-1})\ generate\_rep(a_k))fr)\ ,\end{array}$$

где *generate_rep*(a_r) для каждого r $(1 \le r \le k)$ — результат трансляции a_r на входной язык системы ACL2. Отметим, что любой оператор **c** (в том числе составной) транслируется в одну или две *binding* конструкции. Блок $b*$ в ACL2-коде для составного оператора является объемлющим для этих *binding* конструкций. Таким образом, значение переменной fr при исполнении конструкции $b*$ является значением вектора v при исполнении составного оператора.

Вторая конструкция *binding* определяется как в пункте 4.

Вопрос существования объемлющего блока решается построением дерева вложенности блоков для кода, сгенерированного на втором шаге алгоритма. В качестве корня такого дерева возьмём блок $b*$, сгенерированный функцией *generate_rep_one* на первом шаге алгоритма. Этот блок соответствует телу цикла. Отметим, что каждому оператору тела цикла можно сопоставить блок в данном дереве вложенности блоков. Например, если if-ветвь представлена единственным оператором, то объемлющим для него будет блок $b*$, сгенерированный при трансляции данного if. А если оператор входит в последовательность инструкций составного оператора ветви if, то объемлющим для него будет блок $b*$, полученный при трансляции данного составного оператора. Поиск объемлющего блока для определённого оператора в таком дереве вложенности можно проводить в глубину, пока не будет найден блок $b*$, одна из конструкций *binding* которого будет соответствовать данному оператору. Так как самый объемлющий блок — тело цикла, то такой поиск будет всегда завершаться.

4. Стратегия автоматического доказательства условий корректности в ACL2

При доказательстве условий корректности нашим методом возникает необходимость использования индукции из-за рекурсивного определения операции замены. Хотя ACL2 поддерживает доказательства по индукции, мы столкнулись с определёнными трудностями при проведении экспериментов. Мы предлагаем эвристический метод, позволяющий успешно доказать частичную корректность ряда примеров с финитной итерацией над изменяемыми массивами с выходом из цикла.

Идея состоит в проверке предположения, что постусловие программы описывает случаи в форме конъюнкции импликаций, случился или нет выход из тела цикла. ACL2 способна проверить истинность такого предположения. Если оно истинно, то это помогает более точно описать случаи в постусловии. Доказательство всех уточнённых случаев помогает доказать условие корректности.

На вход алгоритма поступает условие корректности ϕ, переменная n, определение функции rep, теория предметной области и постусловие.

В результате работы алгоритм выдаёт либо «ϕ истина», либо «неизвестно».

Данный алгоритм выглядит следующим образом:

1. Пусть M обозначает кортеж импликаций, являющихся конъюнктами постусловия. Рассмотрим кортеж N, такой что i-й элемент N является посылкой i-го элемента M. Переход на шаг 2.

2. Для каждого элемента из N выполнить шаг 3. Если результат истинен, добавить в теорию лемму, представляющую собой конъюнкцию ϕ и равенства i-го элемента N с $rep(\ldots).$*loop-break*. Иначе перейти на шаг 4. Если результат истинен, добавить в теорию лемму, представляющую собой конъюнкцию ϕ и равенства i-го элемента N с $\neg rep(\ldots).$*loop-break*. Перейти на шаг 5.

3. Пусть θ обозначает i-й элемент N. Пусть ω является конъюнкцией ϕ и равенства θ с $rep(\ldots).$*loop-break*. Проверить истинность ω в ACL2. Если ω была доказана, то вернуть «истина», иначе — «ложь».

4. Пусть θ обозначает i-й элемент N. Пусть ω является конъюнкцией ϕ и равенства θ с $\neg rep(\ldots).$*loop-break*. Проверить истинность ω в ACL2. Если ω была доказана, то вернуть «истина», иначе — «ложь».

5. Проверить истинность ϕ в ACL2 с помощью полученных лемм. Если ϕ была доказана, то вернуть «ϕ истина», иначе — «неизвестно».

Данный алгоритм может быть обобщён для использования с другими интерактивными доказателями и SMT-решателями, например CVC4 и Z3.

5. Эксперимент по автоматической верификации

В данном разделе мы продемонстрируем применение наших методов. Рассмотрим следующую функцию **negate_first** [5], которая в заданном одномерном массиве целых чисел меняет знак первого по порядку индексов отрицательного элемента.

```
void negate_first(int n, int* a) {
     int i;
     for (i = 0; i < n; i++) {
         if (a[i] < 0) {a[i] = -a[i]; break;}}}
```

Предусловие имеет вид:

```
(and (integer-listp a) (integer-listp a_0) (equal a a_0)
     (integerp n) (< 0 n) (<= n (length a_0)))
```

Постусловие выглядит следующим образом:

```
(let (((mv found-spec index-spec) (count_index n a_0)))
     (and (implies (not found-spec) (equal a a_0))
          (implies found-spec (equal a
              (update-nth index-spec (- (nth index-spec a_0)) a_0)))))
```

Определение *count_ index* дано в [8], приложение E. Данная функция возвращает пару: её первый компонент истинен тогда и только тогда, когда массив a содержит отрицательный элемент. В этом случае второй компонент содержит индекс такого элемента.

Структура данных типа $frame$, рекурсивное определение операции замены rep (см. [8], приложение F) и основанное на ней условие корректности были сгенерированы с помощью метода, изложенного в разделе 3. Условие корректности **mytheorem1** является конъюнкцией двух случаев: имеет ли массив отрицательный элемент или нет. Заметим, что постусловие верифицируемой функции также описывает эти два случая. Таким образом мы применили эвристический метод из раздела 4. В итоге была сгенерирована лемма об эквивалентности утверждения о выходе из цикла и утверждения о существовании отрицательного элемента в массиве. Условие корректности (вместе с данной леммой) имеет вид:

```
(defrule my-theorem1 (b* (((mv found-spec index-spec)
  (count_index n a_0)) ((frame fr) (rep n (frame-init a)))) (implies
  (and (integer-listp a) (integer-listp a_0) (equal a a_0) (integerp n)
  (< 0 n) (<= n (length a_0))) (and (implies (not found-spec)
  (equal fr.a a_0)) (implies found-spec (equal fr.a
  (update-nth index-spec (- (nth index-spec a_0)) a_0))))))
  :prep-lemmas ((defrule lemma (b* (((mv found-spec index-spec)
    (count_index n a_0)) ((frame fr) (rep n (frame-init a)))) (implies
    (and (integer-listp a) (integer-listp a_0) (equal a a_0)
    (integerp n) (< 0 n) (<= n (length a_0))) (and (equal fr.loop-break
     found-spec) (implies (not found-spec) (equal fr.a a_0)) (implies
     found-spec (equal fr.a (update-nth index-spec (- (nth index-spec
     a_0)) a_0))))))
  :enable (frame-init count_index rep)
  :induct (dec-induct n))))
```

Система ACL2 успешно доказала условие корректности в полученной теории.

Заключение

Данная статья представляет расширение системы верификации C-light программ [13]. В случае финитной итерации над изменяемыми массивами с выходом из цикла данное расширение позволяет порождать условия корректности без инвариантов циклов.

Эта генерация основана на правиле вывода для оператора **for** языка C-light, использующем операцию замены. Она выражает финитную итерацию в символическом виде. Операция замены порождается автоматически специальным алгоритмом.

Полученные условия корректности автоматически доказываются в ACL2 с помощью предложенного эвристического метода.

Отметим, что верификация функций, реализующих интерфейс BLAS [3], является известной проблемой. Ранее мы успешно выполнили такие эксперименты [7]. Наши методы позволили верифицировать функцию *asum*, реализующую соответствующую функцию из интерфейса BLAS, которая вычисляет сумму абсолютных значений вектора.

В дальнейшем мы планируем рассмотреть случаи более сложных структур данных и верифицировать другие функции интерфейса BLAS.

Список литературы / References

[1] Anureev I. S., Maryasov I. V., Nepomniaschy V. A., “C-programs Verification Based on Mixed Axiomatic Semantics”, *Automatic Control and Computer Sciences*, **45**:7 (2011), 485–500.

[2] Cohen E., Dahlweid M., Hillebrand M., Leinenbach D., Moskal M., Santen T., Schulte W., Tobies S., “VCC: A Practical System for Verifying Concurrent C”, 22nd Int. Conf. TPHOLs, *LNCS*, **5674** (2009), 23–42.

[3] Dongarra J. J., van der Steen A. J., “High-performance computing systems: Status and outlook”, *Acta Numerica*, **21** (2012), 379–474.

[4] Filliâtre J.-C., Marché C., “Multi-prover Verification of C Programs”, 6th ICFEM, *LNCS*, **3308** (2004), 15–29.

[5] Jacobs B., Kiniry J. L., Warnier M., “Java Program Verification Challenges”, FMCO 2002, *LNCS*, **2852** (2003), 202–219.

[6] Kaufmann M., Moore J.S., “An Industrial Strength Theorem Prover for a Logic Based on Common Lisp”, *IEEE Transactions on Software Engineering*, **23**:4 (1997), 203–213.

[7] Kondratyev D., “Implementing the Symbolic Method of Verification in the C-Light Project”, PSI 2017, *LNCS*, **10742** (2018), 227–240.

[8] Kondratyev D.A., “Towards Loop Invariant Elimination for Definite Iterations over Changeable Data Structures in C Programs Verification. Appendices”, https://bitbucket.org/c-light/loop-invariant-elimination.

[9] Kondratyev D.A., Maryasov I.V., Nepomniaschy V.A., “Towards Loop Invariant Elimination for Definite Iterations over Changeable Data Structures in C Programs Verification”, *PSSV 2018*, Workshop Proceedings, Yaroslavl, 2018, 51–57.

[10] Li J., Sun J., Li L., Loc Le Q., Lin S-W., “Automatic Loop Invariant Generation and Refinement through Selective Sampling”, *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, 782–792.

[11] Maryasov I.V., Nepomniaschy V.A., "Loop Invariants Elimination for Definite Iterations over Unchangeable Data Structures in C Programs", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 773–782.

[12] Maryasov I.V., Nepomniaschy V.A., Kondratyev D.A., "Invariant Elimination of Definite Iterations over Arrays in C Programs Verification", *Modeling and Analysis of Information Systems*, **24**:6 (2017), 743–754.

[13] Maryasov I.V., Nepomniaschy V.A., Promsky A.V., Kondratyev D.A., "Automatic C Program Verification Based on Mixed Axiomatic Semantics", *Automatic Control and Computer Sciences*, **48**:7 (2014), 407–414.

[14] Nepomniaschy V.A., "Symbolic Method of Verification of Definite Iterations over Altered Data Structures", *Programming and Computer Software*, **31**:1 (2005), 1–9.

[15] Tuerk T., "Local Reasoning about While-Loops", *VSTTE 2010*, Workshop Proceedings, 2010, 29–39.

Kondratyev D. A.[1], **Maryasov I. V., Nepomnyaschy V. A.**, "The Automation of C Program Verification by Symbolic Method of Loop Invariants Elimination", *Modeling and Analysis of Information Systems*, **25**:5 (2018), 491–505.

DOI: 10.18255/1818-1015-2018-5-491-505

Abstract. During deductive verification of programs written in imperative languages, the generation and proof of verification conditions corresponding to loops can cause difficulties, because each one must be provided with an invariant whose construction is often a challenge. As a rule, the methods of invariant synthesis are heuristic ones. This impedes its application. An alternative is the symbolic method of loop invariant elimination suggested by V.A. Nepomniaschy in 2005. Its idea is to represent a loop body in a form of special replacement operation under certain constraints. This operation expresses loop effect in a symbolic form and allows to introduce an inference rule which uses no invariants in axiomatic semantics. This work represents the further development of this method. It extends the mixed axiomatic semantics method suggested for C-light program verification. This extension includes the verification method of iterations over changeable arrays possibly with loop exit in C-light programs. The method contains the inference rule for iterations without loop invariants. This rule was implemented in verification conditions generator which is a part of the automated system of C-light program verification. To prove verification conditions automatically in ACL2, two algorithms were developed and implemented. The first one automatically generates the replacement operation in ACL2 language, the second one automatically generates auxiliary lemmas which allow to prove the obtained verification conditions in ACL2 successfully in automatic mode. An example which illustrates the application of the mentioned methods is described.

Keywords: C-light, loop invariants, mixed axiomatic semantics, definite iteration, arrays, ACL2, specification, verification, Hoare logic

On the authors:
Dmitry A. Kondratyev, orcid.org/0000-0002-9387-6735, postgraduate student,
A.P. Ershov Institute of Informatics Systems SB RAS,
6 Akademik Lavrentyev av., Novosibirsk 630090, Russia, e-mail: apple-66@mail.ru

Ilya V. Maryasov, orcid.org/0000-0002-2497-6484, PhD,
A.P. Ershov Institute of Informatics Systems SB RAS,
6 Akademik Lavrentyev av., Novosibirsk 630090, Russia, e-mail: ivm@iis.nsk.su

Valery A. Nepomniaschy, orcid.org/0000-0003-1364-5281, PhD,
A.P. Ershov Institute of Informatics Systems SB RAS,
6 Akademik Lavrentyev av., Novosibirsk 630090, Russia, e-mail: vnep@iis.nsk.su

Acknowledgments:
[1] This author is partially supported by RFBR, grant 17-01-00789.

Моделирование и анализ информационных систем. Т. 25, №5 (2018), с. 506–524
Modeling and Analysis of Information Systems. Vol. 25, No 5 (2018), pp. 506–524

Прикладные логики
Applied Logics

DOI: 10.18255/1818-1015-2018-5-506-524

УДК 517.9

О выразительных возможностях некоторых расширений линейной темпоральной логики

Гнатенко А. Р., Захаров В. А.

получена 10 сентября 2018

Аннотация. Конечные автоматы, задающие преобразования потоков входных сигналов в последовательности элементарных действий, являются простейшей моделью вычислений, пригодной для описания поведения реагирующих систем. Это поведение проявляется в соответствии между потоком входных сигналов и последовательностью элементарных действий, выполняемых системой. Мы полагаем, что для формального описания поведения реагирующих систем такого рода требуются более сложные и выразительные средства спецификации, нежели традиционная темпоральная логика линейного времени LTL. В этой работе рассматривается новый язык формальных спецификаций $\mathcal{LP}$-LTL, представляющий собой расширение темпоральной логики LTL, специально разработанное для описания свойств вычислений автоматов-преобразователей. Темпоральные операторы в формулах $\mathcal{LP}$-LTL снабжены параметрами, в качестве которых используются множества слов (языки), описывающие потоки сигналов управления, поступающих на вход реагирующей системы. Базовые предикаты в формулах $\mathcal{LP}$-LTL также определяются языками в алфавите элементарных действий; эти языки описывают ожидаемую реакцию системы в ответ на воздействия окружающей среды. В более ранних работах авторов этой статьи изучалась задача верификации конечных автоматов-преобразователей относительно спецификаций, представленных формулами логик $\mathcal{LP}$-LTL и $\mathcal{LP}$-CTL. Было показано, что для обеих логик эта задача алгоритмически разрешима. Цель данной работы — оценить выразительные возможности логики $\mathcal{LP}$-LTL и сравнить ее с широко известными логиками, применяющимися в информатике для спецификации реагирующих систем. В логике $\mathcal{LP}$-LTL были выделены два фрагмента $\mathcal{LP}$-1-LTL и $\mathcal{LP}$-n-LTL. Нам удалось установить, что язык спецификаций $\mathcal{LP}$-1-LTL более выразителен, чем LTL, в то время как фрагмент $\mathcal{LP}$-n-LTL обладает точно такими же выразительными возможностями, что и монадическая логика второго порядка S1S.

Ключевые слова: темпоральные логики, выразительность, спецификация, верификация, автоматы Бюхи, бесконечные слова

Для цитирования: Гнатенко А. Р., Захаров В. А., "О выразительных возможностях некоторых расширений линейной темпоральной логики", *Моделирование и анализ информационных систем*, **25**:5 (2018), 506–524.

Об авторах:
Гнатенко Антон Романович, orcid.org/0000-0003-1499-2090, студент,
Московский государственный университет им. М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: gnatenko.cmc@gmail.com

Захаров Владимир Анатольевич, orcid.org/0000-0002-3794-9565, доктор физ.-мат. наук, профессор,
Национальный исследовательский университет «Высшая школа экономики»,
ул. Мясницкая, 20, г. Москва, 101000 Россия, e-mail: zakh@cs.msu.ru

Благодарности:
Работа выполнена при финансовой поддержке гранта РФФИ N 18-01-00854

1. Введение

Автоматы-преобразователи являются одной из самых ранних моделей вычислений; они находят применение в информатике, программировании, микроэлектронике, лингвистике. В этой статье мы будем использовать их в качестве формальной модели реагирующих информационных систем. На каждом шаге вычисления автомат-преобразователь получает на входе один из символов входного алфавита и выдает на выходе последовательность символов (слово) выходного алфавита. Буквы входного алфавита соответствуют сигналам управления, поступающим из внешней среды, а буквы выходного алфавита — элементарным действиям, выполняемым реагирующей системой. На множестве элементарных действий, выполняемых автоматом-преобразователем, можно ввести операцию композиции, сформировав, таким образом, полугруппу действий автомата. Поведение системы в этом случае характеризуется отношением преобразования потока управляющих сигналов в действия, выполняемые системой. Модель последовательных реагирующих систем такого вида была предложена и исследована в статьях [6, 21–23].

Но для проверки правильности поведения реагирующих систем, моделируемых конечными автоматами-преобразователями над полугруппами, необходим адекватный этой модели язык формальных спецификаций. Предлагаемый нами подход к построению такого языка основывается на следующих соображениях. Типичное требование правильного поведения реагирующей системы состоит в том, что для каждого входного слова (потока сигналов управления), подпадающего под некоторый заданный шаблон, вырабатывается реакция, которая также подпадает под некоторый заданный шаблон. Для выражения таких требований понадобятся не только темпоральные операторы, при помощи которых можно описывать порядок следования событий (управляющих сигналов и действий), но также и средства для описания и учета указанных шаблонов. Для задания шаблонов потоков управляющих сигналов годятся любые способы описания формальных языков — автоматы, формальные грамматики, регулярные выражения и пр. Для описания реакции автомата-преобразователя, работающего над некоторой заданной полугруппой элементарных действий, можно использовать любые предикаты, определенные в этой полугруппе. В случае свободной полугруппы такие предикаты фактически представляют собой множества слов (языки) над алфавитом элементарных действий. Для их задания также годятся любые средства задания формальных языков. В этом случае свойства поведения реагирующих систем можно описывать, например, при помощи темпоральных формул вида $\mathbf{G}_L P$, означающих, что для каждого входного слова w, принадлежащего языку L, выходное слово h, которое вырабатывает автомат-преобразователь в качестве реакции, принадлежит языку P.

Такой подход обладает двумя достоинствами. Во-первых, он позволяет явно выражать соответствия между входными и выходными словами автоматов-преобразователей и описывать тем самым типичные требования правильного поведения реагирующих систем. И, кроме того, он позволяет адаптировать методы верификации моделей программ, пригодные для традиционных темпоральных логик (см. [2]). В статье [11] этот замысел был осуществлен для языка спецификаций $\mathcal{LP}$-LTL, основанного на темпоральной логике LTL, а затем в статье [7] он был распространен на язык спецификаций $\mathcal{LP}$-CTL, расширяющий логику деревьев вычислений CTL.

Цель данной статьи — оценить выразительные возможности новой темпоральной логики $\mathcal{LP}$-LTL, сравнив ее с другими прикладными логиками, используемыми для верификации программ. Проведение такого сравнения осложняется тем, что семантика $\mathcal{LP}$-LTL и семантика наиболее известных темпоральных логик определяются на разных структурах. Чтобы преодолеть эту трудность, мы выделили в логике $\mathcal{LP}$-LTL два фрагмента $\mathcal{LP}$-1-LTL и $\mathcal{LP}$-n-LTL, семантики которых можно адаптировать к бесконечным словам (сверхсловам), и за счет этого сделать возможным сравнение этих фрагментов, темпоральной логикой линейного времени LTL и монадической логикой второго порядка S1S. В результате удалось обнаружить, что $\mathcal{LP}$-1-LTL является строго более выразительной, нежели LTL, а $\mathcal{LP}$-n-LTL и S1S обладают одинаковыми выразительными способностями.

Статья устроена следующим образом. В разделе 2 определена модель конечного последовательного автомата-преобразователя. В следующем разделе описаны синтаксис и семантика темпоральной логики $\mathcal{LP}$-LTL. В разделе 4 выделены фрагменты $\mathcal{LP}$-1-LTL и $\mathcal{LP}$-n-LTL логики $\mathcal{LP}$-LTL, семантику которых можно определить на сверхсловах. В следующих двух разделах представлены основные результаты этой статьи. В заключении проведено краткое сопоставление введенного нами расширения логики LTL с другими расширениями той же логики и намечены направления дальнейших исследований свойств темпоральной логики $\mathcal{LP}$-LTL.

2. Моделирование реагирующих систем автоматами-преобразователями

Реагирующая система производит вычисление, получая входные сигналы от окружающей среды и выполняя в ответ на них определенные элементарные действия. Рассмотрим конечное множество $\mathcal{C}$ *входных сигналов* и конечное множество $\mathcal{A}$ *элементарных действий*. Конечные слова в алфавите $\mathcal{C}$ мы будем называть *потоками сигналов*, конечные слова в алфавите $\mathcal{A}$ — *составными действиями*, которые можно рассматривать как последовательные композиции элементарных действий. Множества всевозможных потоков сигналов и всевозможных составных действий обозначим записями $\mathcal{C}^*$ и $\mathcal{A}^*$ соответственно. Мы используем стандартные обозначения, принятые в теории формальных языков: запись uv обозначает конкатенацию слов u и v, а знак ε — пустое слово.

Определение 1. Конечным автоматом-преобразователем *над алфавитами* $\mathcal{C}$ *и* $\mathcal{A}$ *называется пятерка* $\Pi = (Q, \mathcal{C}, \mathcal{A}, q_{init}, T)$, *где 1)* Q *— это множество управляющих состояний, 2)* $q_{init} \in Q$ *— начальное состояние, 3)* $T \subseteq Q \times \mathcal{C} \times Q \times \mathcal{A}^*$ *— отношение переходов.*

Мы будем полагать, что отношение переходов T является тотальным, т. е. для каждого состояния q и сигнала управления c существуют такие состояние q' и действие h, для которых выполняется отношение переходов $(q, c, q', h) \in T$. Четвёрка $(q', c, q'', h) \in T$ называется *переходом*: пребывая в состоянии q' и получив сигнал c, автомат переходит в состояние q'' и выполняет составное действие h. Для наглядности будем обозначать переход записью $q' \xrightarrow{c,h} q''$.

Для описания поведения автомата-преобразователя нам понадобятся понятия прогона и траектории. *Прогон* автомата Π — это бесконечная последовательность переходов: $\rho = q_1 \xrightarrow{c_1, h_1} q_2 \xrightarrow{c_2, h_2} \cdots \xrightarrow{c_n, h_n} q_{n+1}, \ldots$ Прогон, начинающийся в состоянии q_{init}, называется *основным.*

Определение 2. *Пусть s_0 — некоторое составное действие, а ρ — указанный выше прогон автомата* Π. *Тогда* траекторией *автомата* Π *на прогоне ρ называется пара $tr = (s_0, \alpha)$, где последовательность $\alpha = (c_1, s_1), (c_2, s_2), \ldots, (c_i, s_i), \ldots$, удовлетворяет условию $s_i = s_{i-1}h_i$ для каждого натурального i.*

Траектория представляет собой возможную реакцию системы, ранее выполнившей действие s_0, в ответ на поток входных сигналов $w = c_1c_2 \ldots c_i \ldots$. В случае, если прогон ρ является основным прогоном автомата, а $s_0 = \varepsilon$, то соответствующая траектория также называется *основной.* Множество всех основных траекторий автомата Π будем обозначать с помощью $Tr(\Pi)$. Под записью $tr|^i$ понимается траектория, представляющая собой «хвост» траектории tr, т. е. траектория $(s_i, \alpha|^i)$, где $\alpha|^i = (c_{i+1}, s_{i+1}), (c_{i+2}, s_{i+2}), \ldots$

3. Язык спецификаций $\mathcal{LP}$-LTL

Для проверки правильности поведения реагирующих систем необходимо иметь формальный язык спецификаций, на котором записываются требования правильности. Поведение реагирующей системы представляет собой процесс, протекающий во времени и проявляющийся в соответствии между потоками сигналов управления, поступающими на вход системы, и действиями, выполняемыми ею. Для спецификации поведения такого рода обычно используются те или иные разновидности темпоральных логик. При выборе подходящей темпоральной логики для формального описания поведения автомата-преобразователя нужно иметь в виду следующие две отличительные особенности предлагаемой модели реагирующих систем:

1. результат работы автомата-преобразователя — это последовательность выполненных им действий из множества $\mathcal{A}$; поэтому атомарные формулы (базовые предикаты) должны интерпретироваться на множестве $\mathcal{A}^*$,

2. поведение автомата зависит не только от течения времени, но также и от потока входных управляющих сигналов; поэтому темпоральные операторы следует *параметризовать*, то есть снабдить описанием тех потоков управляющих сигналов, на которых проверяется поведение автомата.

Чтобы придать языку спецификаций поведения реагирующих систем указанные свойства, авторы [11] предложили новое расширение темпоральной логики линейного времени LTL. В основу этого расширения положен следующий замысел. Проверку правильности реакций автомата-преобразователя необходимо уметь проводить на потоках сигналов управления. Для стороннего наблюдателя поведение автомата на заданном потоке сигналов управления полностью определяется траекториями прогонов автомата на этом потоке. Событиями, доступными наблюдению, являются действия, выполняемые автоматом по ходу прогонов. Правильность поведения

автомата проявляется в том, что эти события происходят в некотором желаемом порядке. Для описания порядка осуществления событий можно привлечь формулы LTL, в которых те или иные типы событий выступают в роли элементарных высказываний (базовых предикатов). При этом множества потоков сигналов управления, на которых проверяется требование правильности поведения автомата, указываются в качестве параметров темпоральных операторов. Таким образом, образуется параметризованное расширение LTL, в котором отношение выполнимости формул определяется на траекториях автоматов-преобразователей. Более подробное описание этого расширения таково.

Любое множество потоков сигналов управления будем рассматривать как язык в алфавите $\mathcal{C}$. Выделим некоторое отдельное семейство языков $\mathcal{L}$ в алфавите $\mathcal{C}$ (например, регулярные языки) и будем называть всякий язык из этого семейства *шаблоном поведения окружения*. Шаблоны поведения призваны описывать допустимые воздействия внешней среды на реагирующую систему.

Составные действия, выполняемые системой в ответ на сигналы управления, также являются словами в алфавите $\mathcal{A}$. Выделим некоторый класс языков $\mathcal{P}$ в алфавите $\mathcal{A}$ и будем называть всякий язык из этого класса *базовым предикатом*. Каждый базовый предикат можно рассматривать как некоторый тип наблюдаемых событий-откликов реагирующей системы в ответ на воздействие внешнего окружения. При использовании шаблонов поведения окружения и базовых предикатов в логических формулах мы предполагаем, что соответствующие языки определены каким-либо конструктивным образом (например, с помощью автоматов, грамматик, машин Тьюринга, и т. п.).

Определение 3. *Пусть задано некоторое семейство $\mathcal{L}$ шаблонов поведения окружения и класс $\mathcal{P}$ базовых предикатов, допускающих конструктивное описание. Множество формул логики $\mathcal{LP}$-LTL — это наименьшее множество выражений, которые могут быть построены по следующим правилам:*

1. *каждый базовый предикат $P \in \mathcal{P}$ является формулой;*
2. *если φ_1, φ_2 — формулы, то выражения $\neg\varphi_1$ и $\varphi_1 \wedge \varphi_2$ являются формулами;*
3. *если φ_1 и φ_2 — формулы, $c \in \mathcal{C}$, и $L \in \mathcal{L}$, то выражения $\mathbf{X}_c\varphi_1$, $\mathbf{Y}_c\varphi_1$, $\mathbf{F}_L\varphi_1$, $\mathbf{G}_L\varphi_1$ и $\varphi_1\,\mathbf{U}_L\varphi_2$ являются формулами.*

Семантика $\mathcal{LP}$-LTL определяется на основе отношения выполнимости формул на интерпретациях, в роли которых выступают траектории автоматов-преобразователей. Предположим, что имеется некоторый автомат Π и его траектория $tr = (s_0, \alpha)$, где $\alpha = (c_1, s_1), (c_2, s_2), \ldots, (c_i, s_i), \ldots$ Тогда для любой формулы ψ запись $tr \models \psi$ будет означать, что формула ψ выполняется на траектории tr автомата Π.

Определение 4. *Отношение выполнимости $\models$ в логике $\mathcal{LP}$-LTL определим индукцией по глубине формулы:*

1. *если P — базовый предикат, то $tr \models P \iff s_0 \in P$;*
2. *$tr \models \neg\varphi \iff$ неверно, что $tr \models \varphi$;*
3. *$tr \models \varphi_1 \wedge \varphi_2 \iff tr \models \varphi_1$ и $tr \models \varphi_2$;*

4. $tr \models \mathbf{X}_c\varphi \iff c = c_1$ *и* $tr|^1 \models \varphi$;

5. $tr \models \mathbf{Y}_c\varphi \iff c \neq c_1$ *или* $tr|^1 \models \varphi$;

6. $tr \models \mathbf{F}_L\varphi \iff \exists i \geqslant 0\colon c_1c_2\ldots c_i \in L$ *и* $tr|^i \models \varphi$;

7. $tr \models \mathbf{G}_L\varphi \iff \forall i \geqslant 0\colon$ *если* $c_1c_2\ldots c_i \in L$, *то* $tr|^i \models \varphi$;

8. $tr \models \varphi\,\mathbf{U}_L\psi \iff \exists i \geqslant 0\colon c_1c_2\ldots c_i \in L$, *такое, что* $tr|^i \models \psi$ *и* $\forall j,\ 0 \leqslant j < i$, *если* $c_1c_2\ldots c_i \in L$, *то* $tr|^j \models \varphi$.

Впервые логика $\mathcal{LP}$-LTL была введена в статье [11]. В отличие от LTL, в этой логике возникает необходимость использовать два разных оператора обращения к следующему моменту времени (NextTime) $\mathbf{X}_c$ и $\mathbf{Y}_c$ в зависимости от того, насколько обязательным является появление в траектории в текущий момент времени управляющего сигнала c, используемого в качестве параметра. Операторы $\mathbf{X}_c$ и $\mathbf{Y}_c$ двойственны друг другу. Двойственными являются также и операторы $\mathbf{F}_L$ и $\mathbf{G}_L$. С помощью отрицания $\neg$ и конъюнкции $\wedge$ можно выразить другие булевы связки, такие как $\vee, \rightarrow, \equiv$. Подобно тому, как это было сделано в приведенном выше определении для оператора $\mathbf{U}$ (Until), в логике $\mathcal{LP}$-LTL можно ввести параметризованные аналоги других темпоральных операторов LTL, например $\mathbf{R}$ (Release) или $\mathbf{W}$ (Weak Until). В статье [11] показано, что многие полезные соотношения равносильности между формулами LTL, наподобие тождеств неподвижной точки, остаются справедливыми и для параметризованных аналогов этих тождеств.

В логике $\mathcal{LP}$-LTL задача верификации автоматов-преобразователей имеет следующую постановку: для произвольного заданного автомата-преобразователя Π и формулы φ проверить, выполняется ли отношение $tr \models \varphi$ для каждой основной траектории tr из множества $Tr(\Pi)$. В статье [11] показано, что эта задача разрешима за дважды экспоненциальное время в том случае, когда семейства $\mathcal{L}$ и $\mathcal{P}$ являются классами регулярных языков и при этом шаблоны поведения окружения и базовые предикаты в формулах $\mathcal{LP}$-LTL описываются с помощью детерминированных конечных автоматов-распознавателей.

В статье [7] идея параметризации темпоральных операторов шаблонами поведения окружения была распространена на логику деревьев вычислений CTL. В этой статье были определены синтаксис и семантика логики $\mathcal{LP}$-CTL и показано, что задача верификации автоматов-преобразователей относительно спецификаций их поведения, представленных формулами $\mathcal{LP}$-CTL над регулярными семействами языков $\mathcal{L}$ и $\mathcal{P}$, разрешима за экспоненциальное время.

Можно предложить и более изощренную интерпретацию действий, выполняемых автоматами-преобразователя, нежели простое формирование выходных слов. Например, эти действия можно рассматривать как элементы некоторой полугруппы или группы с разрешимой проблемой тождеств (чтобы можно было эффективно сравнивать результаты выполнения действий). В этом случае базовые предикаты можно задавать специальными алгебраическими средствами, характерными для того или иного вида полугрупп. Задача верификации автоматов-преобразователей может тогда существенно усложниться: в статье [6] установлено, например, что задача верификации автомата, работающего над свободной коммутативной полугруппой, алгоритмически неразрешима.

4. Два фрагмента логики $\mathcal{LP}$-LTL

Существует немало логик, которые используются для описания поведения вычислительных систем. В этих логиках каждая формула является описанием множества всех тех вычислений-интерпретаций, на которых она выполняется. Таким образом, возникает возможность сравнивать выразительные способности различных логик в зависимости от того, какие множества вычислений могут быть описаны формулами этих логик. Более строго отношение сравнения выразительных возможностей логик определяется следующим образом.

Определение 5. *Пусть имеются две логики $\mathscr{L}_1$ и $\mathscr{L}_2$, у которых отношение выполнимости формул определяется в одном и том же классе интерпретаций. Будем говорить, что формула ψ_1 одной из этих логик* эквивалентна *формуле ψ_2 какой-либо из этих логик, если соотношение $I \models \psi_1 \Leftrightarrow I \models \psi_2$ верно для каждой интерпретации I. Логика $\mathscr{L}_1$ считается* не менее выразительной, *чем логика $\mathscr{L}_2$ (обозначается $\mathscr{L}_2 \preccurlyeq \mathscr{L}_1$), если для любой формулы логики $\mathscr{L}_2$ существует эквивалентная ей формула логики $\mathscr{L}_2$. Логики $\mathscr{L}_1$ и $\mathscr{L}_2$ имеют* равные выразительные возможности *(обозначается $\mathscr{L}_2 \equiv \mathscr{L}_1$), если верны соотношения $\mathscr{L}_2 \preccurlyeq \mathscr{L}_1$ и $\mathscr{L}_1 \preccurlyeq \mathscr{L}_2$. Логика $\mathscr{L}_1$ считается* более выразительной, *чем логика $\mathscr{L}_2$ (обозначается $\mathscr{L}_2 \prec \mathscr{L}_1$), если $\mathscr{L}_2 \preccurlyeq \mathscr{L}_1$, но при этом и $\mathscr{L}_2 \not\equiv \mathscr{L}_1$.*

Размеченные системы переходов широко используются в качестве моделей реагирующих систем. Наблюдаемое поведение таких систем представляется бесконечной последовательностью наблюдаемых событий. На множестве таких последовательностей можно определить семантику трех логик — теории линейного порядка $(\mathbb{N}, <)$, темпоральной логики линейного времени LTL и монадической логики второго порядка с одной функцией следования S1S. В работе [10] было установлено, что логики $(\mathbb{N}, <)$ и LTL имеют равные выразительные возможности, а в статьях [17, 19] было показано, что логика S1S является более выразительной, чем LTL.

Автоматы-преобразователи также являются моделями реагирующих систем, а их поведение можно описывать посредством нового расширения логики линейного времени $\mathcal{LP}$-LTL. Цель данной статьи — сравнить выразительные возможности трех логик, применяемых для описания поведения реагирующих систем — LTL, S1S и $\mathcal{LP}$-LTL. Однако отношение выполнимости для формул $\mathcal{LP}$-LTL определяется на интерпретациях, которые отличаются от тех, что используются в определениях семантики логик LTL и S1S. Чтобы сравнивать $\mathcal{LP}$-LTL с LTL и S1S, мы выделим два фрагмента $\mathcal{LP}$-LTL, семантику которых можно равносильным образом определить на бесконечных последовательностях событий, и сравним выразительные возможности этих фрагментов с описательными способностями логик LTL и S1S.

Формулы первого из этих фрагментов $\mathcal{LP}$-1-LTL строятся над однобуквенным алфавитом $\mathcal{C} = \{c\}$ входных сигналов и произвольным конечным алфавитом элементарных действий $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$. В качестве семейства $\mathcal{L}$ шаблонов поведения окружения в формулах этого фрагмента разрешается использовать любые регулярные языки над алфавитом $\{c\}$, а в качестве семейства $\mathcal{P}$ базовых предикатов будем использовать набор из n регулярных языков вида $P_i = \mathcal{A}^* a_i$. Будем говорить, что сверхслово $w = a_{i_1} a_{i_2} \ldots a_{i_m} \ldots$ удовлетворяет формуле φ' из фрагмента $\mathcal{LP}$-1-LTL тогда и только тогда, когда для траектории $tr' = (\varepsilon, \alpha'_w)$, где $\alpha'_w = (c, a_{i_1}), (c, a_{i_1} a_{i_2}), \ldots, (c, a_{i_1} a_{i_2} \ldots a_{i_m}), \ldots$, верно, что $tr' \models \varphi'$.

Фрагмент $\mathcal{LP}$-n-LTL определяется аналогично, с той лишь разницей, что алфавиты $\mathcal{C}$ и $\mathcal{A}$ совпадают, т. е. $\mathcal{C} = \mathcal{A} = \{a_1, a_2, \ldots, a_n\}$. Сверхслово $w = a_{i_1}a_{i_2}\ldots a_{i_m}\ldots$ удовлетворяет формуле φ'' из фрагмента $\mathcal{LP}$-n-LTL тогда и только тогда, когда для траектории $tr'' = (\varepsilon, \alpha''_w)$, где $\alpha''_w = (a_{i_1}, a_{i_1}), (a_{i_2}, a_{i_1}a_{i_2}), \ldots, (a_{i_m}, a_{i_1}a_{i_2}\ldots a_{i_m}), \ldots$, верно, что $tr'' \models \varphi''$.

Чтобы сравнивать выразительные возможности логик LTL, S1S и двух выделенных фрагментов логики $\mathcal{LP}$-LTL, приведем альтернативное определение семантики формул из рассмотренных фрагментов на сверхсловах. Пусть $\Sigma = \{a_1, a_2, \ldots, a_n\}$ — некоторый конечный алфавит. *Сверхсловом* будем называть всякое отображение $w\colon \mathbb{N} \to \Sigma$, где $\mathbb{N}$ обозначает множество неотрицательных целых чисел. Множество всех сверхслов обозначим записью Σ^ω. Запись $w|^k$ будет обозначать суффикс слова w, начинающийся с буквы с номером k, т. е. бесконечного слова v, для которого $v(i) = w(i+k)$ при любом i, $i \geqslant 0$. Запись $w[0\ldots k]$ будет обозначать префикс w, т. е. конечное слово $w(0)w(1)\ldots w(k)$. В случае, когда L является регулярным языком над однобуквенным алфавитом $\{c\}$, будем вместо $c^i \in L$ писать кратко $i \in L$.

Переформулируем определение семантики формул фрагмента $\mathcal{LP}$-1-LTL как множества формул, построенных из букв алфавита Σ (используемых вместо базовых предикатов) с помощью булевых связок $\neg, \wedge$ и темпоральных операторов $\mathbf{X}$, $\mathbf{F}_L$, $\mathbf{G}_L$ и $\mathbf{U}_L$, параметризованных регулярными языками L над однобуквенным алфавитом $\mathcal{C} = \{c\}$.

Определение 6. *Отношение выполнимости $\models$ формул $\mathcal{LP}$-1-LTL на множестве сверхслов определяется следующими правилами:*

1. *для атомарной формулы $a \in \Sigma$: $w \models a \iff w(0) = a$;*
2. $w \models \neg\varphi \iff$ *неверно, что* $w \models \varphi$;
3. $w \models \varphi \wedge \psi \iff w \models \varphi$ *и* $w \models \psi$;
4. $w \models \mathbf{X}\varphi \iff w|^1 \models \varphi$;
5. $w \models \mathbf{F}_L\varphi \iff \exists\, i \geqslant 0$, *такое, что* $i \in L$ *и* $w|^i \models \varphi$;
6. $w \models \mathbf{G}_L\varphi \iff \forall\, i \geqslant 0$, *если* $i \in L$, *то* $w|^i \models \varphi$;
7. $w \models \varphi\, \mathbf{U}_L\psi \iff \exists\, i \geqslant 0$, *такое, что* $i \in L$ *и* $w|^i \models \psi$, *и* $\forall\, j$, $0 \leqslant j < i$, *если* $j \in L$, *то* $w|^j \models \varphi$.

Примером формулы фрагмента $\mathcal{LP}$-1-LTL может служить выражение $\mathbf{G}_{(cc)^*}a$. Как можно понять из определения, сверхслово w удовлетворяет данной формуле тогда и только тогда, когда для любого i, $i \geqslant 0$, верно $w(2i) = a$.

Синтаксис $\mathcal{LP}$-n-LTL отличается от случая $\mathcal{LP}$-1-LTL в трех аспектах:

1. помимо оператора $\mathbf{X}$ ("NeXttime" , в следующий момент времени) появляется двойственный ему оператор $\mathbf{Y}$;
2. операторы $\mathbf{X}$ и $\mathbf{Y}$ параметризованы буквами алфавита Σ: $\mathbf{X}_a$, $\mathbf{Y}_b$;
3. операторы $\mathbf{F}_L$, $\mathbf{G}_L$ и $\mathbf{U}_L$ параметризованы регулярными языками над алфавитом Σ.

Определение 7. *Отношение выполнимости* $\models$ *формул* $\mathcal{LP}$*-n-LTL на сверхсловах определяется следующим образом:*

1. *выполнимость атомарных формул, отрицания* $\neg$ *и конъюнкции* $\wedge$ *определяются точно так же, как в случае* $\mathcal{LP}$-1-LTL*;*
2. $w \models \mathbf{X}_c\varphi \iff w(0) = c$ *и* $w|^1 \models \varphi$*;*
3. $w \models \mathbf{Y}_c\varphi \iff w(0) \neq c$ *или* $w|^1 \models \varphi$*;*
4. $w \models \mathbf{F}_L\varphi \iff \exists\, i \geqslant 0$*, такое, что* $w[0 \ldots i] \in L$ *и* $w|^i \models \varphi$*;*
5. $w \models \mathbf{G}_L\varphi \iff \forall\, i \geqslant 0$*, если* $w[0 \ldots i] \in L$*, то* $w|^i \models \varphi$*;*
6. $w \models \varphi\, \mathbf{U}_L\psi \iff \exists\, i \geqslant 0$*, такое, что* $w[0 \ldots i] \in L$ *и* $w|^i \models \psi$*, и* $\forall\, j,\ 0 \leqslant j < i$*, если* $w[0 \ldots j] \in L$*, то* $w|^j \models \varphi$*.*

5. $\mathcal{LP}$-1-LTL vs LTL

Легко заметить, что темпоральная логика LTL является подмножеством $\mathcal{LP}$-1-LTL, в котором в качестве параметра всех темпоральных операторов используется регулярный язык $\mathcal{C}^*$. Отсюда следует, что $LTL \preccurlyeq \mathcal{LP}$-1-$LTL$. Более того, справедливо следующее более сильное утверждение.

Теорема 1. $LTL \prec \mathcal{LP}$-1-LTL

В статье [19] показано, что формулами логики LTL невозможно описать множество бесконечных слов, имеющих заданную букву во всех позициях, номер которых кратен некоторому числу $k, k \geq 2$. Мы приводим здесь другое доказательство этого результата с помощью *игры Эренфойхта–Фреше* (см. [3]).

Всюду далее в формулах LTL вместо оператора $\mathbf{G}$ будем использовать эквивалентную комбинацию $\neg\, \mathbf{F}\, \neg$. Техника *игр Эренфойхта–Фреше* для темпоральных логик была введена авторами статьи [3] для построения иерархии формул LTL. Для описания этой техники нам понадобится понятие *глубины вложенности операторов* формулы.

Определение 8. *Определим* глубину вложенности операторов $depth(\varphi)$ *формулы* $\varphi \in LTL$ *индукцией по ее строению:*

1. *для атомарной формулы* $a \in \Sigma$ *считается, что* $depth(a) = 0$*;*
2. $depth(\neg\varphi) = depth(\varphi)$*;*
3. $depth(\varphi \wedge \psi) = \max(depth(\varphi), depth(\psi))$*;*
4. $depth(\mathbf{X}\varphi) = depth(\varphi) + 1$*;*
5. $depth(\mathbf{F}\varphi) = depth(\varphi) + 1$*;*
6. $depth(\varphi\mathbf{U}\psi) = \max(depth(\varphi), depth(\psi)) + 1$.

В игре Эренфойхта–Фреше принимают участие два игрока: Игрок 1 и Игрок 2; один из них в результате выигрывает, а другой проигрывает. Игра ведется на паре сверхслов (w_0, w_1), которая называется *конфигурацией* игры. Игрок 1 стремится показать, что сверхслова w_0 и w_1 *различимы* некоторой формулой LTL, в то время как Игрок 2 пытается доказать обратное.

Определение 9. *Игра Эренфойхта–Фреше, состоящая из* k раундов *(*k*-игра) с начальной конфигурацией* (w_0, w_1)*, определяется индукцией по* k*:*

- *Игрок 1 выигрывает в игру из 0 раундов, если* $w_0(0) \neq w_1(0)$*. В противном случае выигрывает Игрок 2.*
- *Игрок 2 выигрывает в игру из* $(k{+}1)$ *раундов, если* $w_0(0) \neq w_1(0)$*. В противном случае разыгрывается раунд, что приводит либо к победе Игрока 1, либо к новой конфигурации* (w_0', w_1')*, на которой разыгрывается игра из* k *раундов.*

В каждом *раунде* проводится изменение конфигураций. Игрок 1 выбирает один из трех возможных *ходов*:

- **X-ход.** В качестве новой конфигурации устанавливает пару $(w_0|^1, w_1|^1)$;
- **F-ход.**
  1. Игрок 1 выбирает слово w_s, $s \in \{0, 1\}$, и позицию $i_s \geqslant 0$;
  2. Игрок 2 отвечает, выбирая позицию $i_{1-s} \geqslant 0$ в слове w_{1-s};
  3. В качестве новой конфигурации устанавливается пара $(w_s|^{i_s}, w_{1-s}|^{i_{1-s}})$.
- **U-ход.**
  1. Игрок 1 выбирает w_s, $s \in \{0, 1\}$, и позицию $i \geqslant 0$;
  2. Игрок 2 отвечает, выбирая такую позицию $i_{1-s} \geqslant 0$ в слове w_{1-s}, что если $i_s = 0$, то $i_{1-s} = 0$;
  3. Игрок 1 выбирает один из двух вариантов развития событий:
     (a) В качестве новой конфигурации устанавливает пару $(w_s|^{i_s}, w_{1-s}|^{i_{1-s}})$;
     (b) Игрок 1 выбирает позицию i'_{1-s} в слове w_{1-s}, где $0 \leqslant i'_{1-s} < i_{1-s}$, и Игрок 2 отвечает, выбирая позицию $i'_s \in w_s$, $0 \leqslant i'_s < i_s$. В качестве новой конфигурации устанавливается пара $(w_s|^{i'_s}, w_{1-s}|^{i'_{1-s}})$.

Говорят, что у Игрока 2 есть *выигрышная стратегия* в игре, состоящей из k раундов, с начальной конфигурацией (w_0, w_1), если он может выиграть в этой игре независимо от ходов Игрока 1. Этот факт обозначается записью $w_0 \sim_k w_1$. В противном случае говорят, что выигрышную стратегию имеет Игрок 1.

В [3] доказано важное свойство игр Эренфойхта–Фреше:

Утверждение 1. *Для любого целого неотрицательного* k *и любых двух сверхслов* w_0 *и* w_1 *отношение* $w_0 \sim_k w_1$ *верно в том и только в том случае, если для любой формулы* $\varphi \in LTL$ *глубины* $depth(\varphi) \leqslant k$ *верно, что* $w_0 \models \varphi \iff w_1 \models \varphi$*.*

Теперь можно приступить к доказательству теоремы 1. Покажем, что свойство $W_a^{2n} = \{w \in \Sigma^\omega \mid w(2i) = a,\ i \in \mathbb{N}_0\}$ сверхслов в алфавите $\Sigma = \{a, b\}$, описываемое $\mathcal{LP}$-1-LTL формулой $\mathbf{G}_{(cc)^*}a$, невыразимо формулами логики LTL.

Доказательство. Рассмотрим пару слов $u_n = a^{2n+1}ba^\omega \in W_a^{2n}$ и $v_n = a^{2n}ba^\omega \notin W_a^{2n}$. Убедимся, используя технику игр Эренфойхта–Фреше, что при $2n > k$ формулы LTL не различают u_n и v_n. Для удобства префиксы слов u_n и v_n до буквы b будем называть их *началами*, а после буквы b — *хвостами*. В основе предлагаемого строгого доказательства лежит идея, заключающаяся в том, что Игроку 1 не удастся различить блоки a^{2n+1} и a^{2n} за k раундов, и поэтому он не сможет установить разницу между u_n и v_n.

Для строгого обоснования опишем выигрышную стратегию Игрока 2:

1. Если Игрок 1 делает **X**-ход, то Игрок 2 должен играть в соответствии с правилами (происходит "сдвиг" конфигурации на одну букву, в то время как число раундов уменьшается на единицу);

2. Если Игрок 1 делает **F**-ход

   (a) в *хвост*, то Игрок 2 должен тоже сделать ход в *хвост* на ту же самую позицию (считая от первой буквы *хвоста*);

   (b) в *начало*, то Игрок 2 должен сделать ход в *начало*, так, чтобы расстояние от новой позиции до последней буквы *начала* было таким же, как в слове Игрока 1.

   (c) в *позицию буквы* b, то Игрок 2 также должен ходить в *позицию буквы* b;

3. Если Игрок 1 делает **U**-ход, то Игроку 2 следует играть аналогично случаю **F**-хода, за исключением ситуации, когда Игрок 1 делает ход в первую букву *начала* (т. е. остается в предыдущей позиции); тогда Игрок 2 должен играть согласно правилам и тоже сделать ход в первую букву *начала*.

   Аналогично, если затем Игрок 1 сделает "ход назад" (применив третий пункт описания **U**-хода), то Игроку 2 следует отвечать на его действия, как описано выше, за исключением случая, когда Игрок 1 делает ход в первую букву *начала* (тогда Игроку 2 следует также делать ход в первую букву *начала*).

Легко заметить, что справедливо следующее

Утверждение 2. *При использовании Игроком 2 описанной выше стратегии конфигурация* (u, v) *на всем протяжении игры удовлетворяет следующим свойствам:*

1. $u[0] = v[0]$*;*

2. *если* $u \neq v$*, то длины кратчайших блоков букв* a *в словах* u *и* v *не меньше, чем оставшееся количество раундов;*

3. *между второй и третьей частями* **U***-хода для любой позиции* j *в слове* w_2 *Игрока 2, которая меньше текущей позиции, существует соответствующая ей позиция* i *в слове* w_1 *Игрока 1, такая, что конфигурация* $(w_1|^i, w_2|^j)$ *удовлетворяет свойствам 1 и 2.*

Таким образом, теорема 1 доказана. □

6. $\mathcal{LP}$-n-LTL vs S1S

Перейдем теперь к исследованию выразительных возможностей фрагмента $\mathcal{LP}$-n-LTL. Очевидно, что фрагмент $\mathcal{LP}$-n-LTL является не менее выразительным, чем фрагмент $\mathcal{LP}$-1-LTL. Далее мы покажем, что он столь же выразителен, как монадическая логика второго порядка S1S. Для доказательства этого факта мы не будем рассматривать синтаксис и семантику логики S1S, а воспользуемся вместо этого взаимосвязью этой логики с конечными автоматами, предназначенными для распознавания сверхслов. Хорошо известно (см., например, [15]), что следующие три утверждения о языках сверхслов эквивалентны:

- множество L состоит из всех тех сверхслов, на которых выполняется некоторая формула Φ логики S1S, т. е. $L = \{w \mid w \models \Phi\}$,
- множество сверхслов L является *ω-регулярным языком*,
- множество сверхслов LL распознаётся некоторым *автоматом Бюхи (Рабина, Маллера, Стрита)*.

Поэтому мы будем сравнивать выразительные возможности фрагмента $\mathcal{LP}$-n-LTL с вычислительными способностями конечных ω-автоматов.

Определение 10. Конечным автоматом над сверхсловами *(ω-автоматом) называется пятёрка* $A = (\Sigma, Q, Q_0, \Delta, \mathcal{F})$, *где*

- Σ — *это конечный алфавит*,
- Q — *это конечное множество* состояний,
- $\Delta \subseteq Q \times \Sigma \times Q$ — отношение переходов,
- $Q_0 \in Q$ — *множество* начальных состояний,
- $\mathcal{F}$ — допускающее правило.

Детерминированным называется такой ω-автомат, у которого $Q_0 = \{q_0\}$ и отношение переходов является функцией, т. е. для любого состояния q и любой буквы a существует единственное состояние $q' \in Q$, для которого $(q, a, q') \in \Delta$. *Прогоном* ω-автомата A на сверхслове w называется всякое такое отображение $\rho\colon \mathbb{N}_0 \to Q$, у которого $\rho(0) \in Q_0$, и для каждого i выполняется соотношение $(\rho(i), w(i), \rho(i+1)) \in \Delta$.

Записью $\inf(\rho)$ обозначим множество состояний, которые встречаются в прогоне ρ бесконечно часто. Разнообразные виды ω-автоматов отличаются друг от друга допускающими правилами. Здесь мы рассматриваем автоматы Бюхи и автоматы Маллера. Допускающее правило обобщенного автомата Бюхи B представляет собой семейство $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$ подмножеств F_i, $F_i \subseteq Q$, $1 \leqslant i \leqslant m$, множества состояний автомата B.

Определение 11. *Автомат Бюхи B* допускает *сверхслово w тогда и только тогда, когда существует такой прогон ρ этого автомата на сверхслове w, что* $\inf(\rho) \cap F_i \neq \varnothing$ *для каждого i*, $1 \leqslant i \leqslant m$.

Допускающее правило автомата Маллера M также задается семейством $\mathcal{F} = \{E_1, E_2, \ldots, E_m\}$ подмножеств E_i, $E_i \subseteq Q$, $1 \leqslant i \leqslant m$, множества состояний ω-автомата M.

Определение 12. *Автомат Маллера M* допускает *сверхслово w тогда и только тогда, когда существует такой прогон ρ этого автомата на сверхслове w, что* $\inf(\rho) \in \mathcal{F}$.

Множество сверхслов, допускаемых автоматом A, обозначим записью $L(A)$. Говорят, что автомат A *распознает* язык $L(A)$.

Покажем, что класс языков, распознаваемых ω-автоматами, совпадает с классом языков, которые можно описать формулами логики $\mathcal{LP}$-n-LTL. Для этого, во-первых, каждой формуле $\psi \in \mathcal{LP}$-n-LTL поставим в соответствие такой недетерминированный автомат Бюхи B_ψ, что $L(B_\psi) = \{w \mid w \models \psi\}$. А во-вторых, для каждого детерминированного автомата Маллера M построим такую формулу $\psi_M \in \mathcal{LP}$-n-LTL, что $L(M) = \{w \mid w \models \psi_M\}$.

В первом случае нам потребуется несколько модифицировать хорошо известную схему построения автомата Бюхи по формуле логики LTL (см., например, [1]). Пусть $\varphi \in \mathcal{LP}$-n-LTL. Перепишем формулу φ, используя тождества $\mathbf{G}_L\psi = \neg\mathbf{F}_L\neg\psi$ и $\mathbf{F}_L\psi = \mathbf{true}\,\mathbf{U}_L\psi$ так, чтобы исключить операторы $\mathbf{F}$ и $\mathbf{G}$, оставив только $\mathbf{U}$. Далее без ограничения общности полагаем, что φ построена с помощью $\neg$, $\wedge$, $\mathbf{X}_c$, $\mathbf{Y}_c$, $\mathbf{U}_L$.

Определение 13. Замыканием $cl(\varphi)$ *формулы φ называется множество всех подформул этой формулы, т. е. наименьшее такое множество, что*

- $\varphi \in cl(\varphi)$;
- *если* $\neg\psi \in cl(\varphi)$, *то* $\psi \in cl(\varphi)$;
- *если* $\psi \wedge \chi \in cl(\varphi)$, *то* $\psi, \chi \in cl(\varphi)$;
- *если* $\mathbf{X}\psi \in cl(\varphi)$, *то* $\psi \in cl(\varphi)$;
- *если* $\psi\,\mathbf{U}_L\chi \in cl(\varphi)$, *то* $\psi, \chi \in cl(\varphi)$.

Сверхслову $w \in \Sigma^\omega$ поставим в соответствие такое сверхслово ρ_w в алфавите $2^{cl(\varphi)}$, что для каждого i включение $\psi \in \rho_w(i)$ имеет место тогда и только тогда, когда $w|^i \models \psi$. Далее мы построим автомат Бюхи $\mathcal{B}_\varphi$, у которого множеством состояний является семейство всех подмножеств замыкания $cl(\varphi)$ формулы φ, а последовательность ρ_w — это прогон B на слове w. С помощью состояний, переходов и допускающего правила мы опишем семантику формулы φ, причем для описания семантики параметров операторов мы воспользуемся конечными автоматами.

Определение 14. *(Неинициализированным)* конечным автоматом *назовем тройку $A = (\Sigma, Q, \delta)$, где Σ — это алфавит, Q — множество состояний, а $\delta\colon Q \times \Sigma \to Q$ — функция переходов.*

Функцию переходов обычным образом распространим на множестве Σ^* всех конечных слов в алфавите Σ: $\delta(q, \varepsilon) = q$ для пустого слова ε, и $\delta(q, \alpha a) = \delta(\delta(q, \alpha), a)$ для любых $a \in \Sigma$ и $\alpha \in \Sigma^*$. Автомат A называется *инициализированным*, если

задано *начальное состояние* q_0 и множество допускающих состояний F. Инициализированный таким образом автомат обозначается $A(q_0, F)$; он *допускает* конечное слово α тогда и только тогда, когда $\delta(q_0, \alpha) \in F$. Записью $L(A(q_0, F))$ мы обозначаем множество всех слов, допускаемых автоматом $A(q_0, F)$, т. е. *язык* этого автомата. Далее, упоминая регулярный язык L, мы полагаем, что он распознается автоматом $A^L({q_0}^L, F^L)$, где $A^L = (\Sigma, Q^L, \delta^L)$.

Утверждение 3. *Для каждой формулы φ логики $\mathcal{LP}$-n-LTL существует обобщенный автомат Бюхи $\mathcal{B}_\varphi$, обладающий следующим свойством: для любого сверхслова w отношение $w \models \varphi$ выполняется тогда и только тогда, когда w допускается автоматом Бюхи $\mathcal{B}_\varphi$.*

Доказательство. Пусть φ — произвольная формула логики $\mathcal{LP}$-n-LTL. Опишем устройство соответствующего обобщенного автомата Бюхи $\mathcal{B}_\varphi$. Для этого построим множество $\widehat{Q} = 2^{cl(\varphi)} \times Q^{L_1} \times Q^{L_2} \times \ldots \times Q^{L_m}$, где $L_1, L_2, \ldots, L_m$ — параметры всех темпоральных операторов, используемых в φ. Важно отметить, что, если некоторые операторы наделены одним и тем же параметром L, этот параметр всё равно включается в список $L_1, L_2, \ldots, L_m$ отдельно для каждого оператора.

Назовём элемент $q = (S, q^{L_1}, \ldots, q^{L_m}) \in \widehat{Q}$ *нормальным*, если

- множество формул S непротиворечиво относительно классической пропозициональной логики, т. е.
  - если $\psi \wedge \chi \in S$, то $\psi \in S$ и $\chi \in S$;
  - если $\psi \in S$, то $\neg\psi \notin S$;
  - если $\mathbf{true} \in cl(\varphi)$, то $\mathbf{true} \in S$;
  - если $a \in S \cap \Sigma$ и $b \in S \cap \Sigma$, то $a = b$;
- q локально непротиворечив относительно оператора $\mathbf{U}$, т. е. для любой формулы вида $\psi\,\mathbf{U_L}\chi \in cl(\varphi)$
  - если $\chi \in S$ и $q^L \in F^L$, то $\psi\,\mathbf{U_L}\chi \in S$;
  - если $\psi\,\mathbf{U_L}\chi \in S$, и $\chi \notin S$, и $q^L \in F^L$, то $\psi \in S$;
- множество формул S максимально, т. е.
  - $S \cap \Sigma \neq \varnothing$;
  - для любой формулы $\psi \in cl(\varphi)$ если $\psi \notin S$, то $\neg\psi \in S$.

Записью $Norm(\varphi)$ обозначим множество всех нормальных элементов $\widehat{Q}$.

Приступим теперь непосредственно к построению автомата Бюхи $\mathcal{B}_\varphi$, распознающего множество сверхслов, на которых выполняется заданная формула φ логики $\mathcal{LP}$-n-LTL. В отличие от аналогичного метода, описанного в монографии [1] для построения автомата Бюхи, соответствующего формуле LTL, нам приходится учитывать конечные автоматы, которые используются для параметризации темпоральных операторов.

Рассмотрим автомат Бюхи $\mathcal{B}_\varphi = (\Sigma, Q, Q_0, \Delta, \mathcal{F})$, где

- $Q = Norm(\varphi)$;
- $Q_0 = \{q \in Q \mid q = (S, q_0^{L_1}, \ldots, q_0^{L_m}),\ \varphi \in S\}$;
- $\mathcal{F} = \{F_{\psi\,\mathbf{U}_L\chi} \mid \psi\,\mathbf{U}_L\chi \in cl(\varphi)\}$, где

$$F_{\psi\,\mathbf{U}_L\chi} = \{q \in Q \mid \psi\,\mathbf{U}_L\chi \notin S \text{ или совместно } \chi \in S \text{ и } q^L \in F^L\}$$

- Отношение переходов $\Delta\colon Q \times \Sigma \to 2^Q$ для состояния $q = (S, q^{L_1}, \ldots, q^{L_m}) \in Q$ и буквы $a \in \Sigma$ определяется следующим образом:
  - если $a \notin S \cap \Sigma$ или существует такая подформула $\mathbf{X}_c\psi \in S$, что $a \neq c$, то $\Delta(q, a) = \varnothing$;
  - если $S \cap \Sigma = \{a\}$ и для каждой подформулы $\mathbf{X}_c\psi \in S$ верно, что $a = c$, то $\Delta(q, a)$ равно множеству всех состояний таких $\hat{q} = (\hat{S}, \hat{q}^{L_1}, \ldots, \hat{q}^{L_m}) \in Q$, для которых выполнены следующие условия:
    1. для каждой подформулы $\mathbf{X}_c\psi \in cl(\varphi)$: $\mathbf{X}_c\psi \in S \iff \psi \in \hat{S}$;
    2. для каждой подформулы $\mathbf{Y}_c\psi \in cl(\varphi)$: если $a = c$, то $\mathbf{Y}_c\psi \in S \iff \psi \in \hat{S}$;
    3. для каждой подформулы $\psi\,\mathbf{U}_L\chi \in cl(\varphi)$: $\psi\,\mathbf{U}_L\chi \in S \iff (q^L \in F^L$ и $\chi \in S)$ или одновременно $\psi\,\mathbf{U}_L\chi \in \hat{S}$ и, если $q^L \in F^L$, то $\psi \in S)$.
    4. для каждого параметра L_i, если помеченная им подформула $\psi\,\mathbf{U}_{L_i}\chi$ принадлежит S, то $\hat{q}^{L_i} = \delta^{L_i}(q^{L_i}, a)$, а иначе $\hat{q}^{L_i} = q^{L_i}$.

По сути дела, ограничения, налагаемые на отношение переходов Δ, полностью описывают семантику темпоральных операторов $\mathbf{X}$, $\mathbf{Y}$ и $\mathbf{U}$. Последнее правило в описании отношения переходов Δ, в частности, гарантирует, что каждый автомат, отвечающий за параметр темпорального оператора, начинает свою работу, как только начинается обработка подформулы, озаглавленной этим оператором.

Чтобы вычисления автомата Бюхи $\mathcal{B}_\varphi$ корректно соответствовали семантике операторов Until $\mathbf{U}_L$, встречающихся в формуле φ, для каждой подформулы $\zeta = \psi\,\mathbf{U}_L\chi$ формулы φ в допускающем правиле для этого ω-автомата имеется множество состояний F_ζ. Присутствие такого множества в допускающем правиле гарантирует, что в каждом прогоне ρ_w, для которого $\zeta \in \rho_w(0)$, будет верно включение $\chi \in S_j$ для некоторого $j \geqslant 0$, $\rho_w(j) = (S_j, \ldots, q_j^L, \ldots)$, $q_j^L \in F^L$ (т. е., $w[0 \ldots j] \in L$); и $\psi \in S_i$ для всех $i < j$, таких, что $q_i^L \in F^L$ (то есть, $w[0 \ldots i] \in L$). В конструкции F_ζ учтено также и влияние на выполнимость оператора Until его параметра: сверхслово w удовлетворяет формуле $\psi\,\mathbf{U}_L\chi$ только в том случае, когда χ в итоге становится верной в позиции, "согласованной" с параметром L.

Далее нетрудно показать, воспользовавшись той же самой схемой рассуждений, которая приведена в монографии [1] для обоснования аналогичного рассуждения, что построенный таким образом автомат Бюхи $\mathcal{B}_\varphi$ обладает требуемым свойством: для любого сверхслова w отношение $w \models \varphi$ выполняется тогда и только тогда, когда w допускается автоматом Бюхи $\mathcal{B}_\varphi$. □

Если принять во внимание сделанное ранее замечание о взаимосвязи автоматов Бюхи и логики S1S, то на основании утверждения 3 можно прийти к заключению о том, что $\mathcal{LP}$-n-$LTL \preccurlyeq$ S1S.

Покажем теперь, как по автомату Маллера построить эквивалентную ему формулу логики $\mathcal{LP}$-n-LTL.

Утверждение 4. *Для каждого детерминированного автомата Маллера существует формула φ логики $\mathcal{LP}$-n-LTL, которая удовлетворяет следующему требованию: автомат M допускает сверхслово w тогда и только тогда, когда $w \models \varphi_M$.*

Доказательство. Пусть задан произвольный детерминированный автомат Маллера $M = (\Sigma, Q, q_0, \Delta, \mathcal{F})$ с единственным начальным состоянием q_0 и допускающим правилом $\mathcal{F} = \{E_1, E_2, \ldots, E_m\}$. Напомним, что *допускающий прогон* автомата M — это такой его прогон ρ, что $\inf(\rho) = E_i$ для некоторого $E_i \in \mathcal{F}$. Заметим, что единственный прогон ρ_w детерминированного автомата Маллера на слове w однозначно определяется этим словом, и состояние, в котором находится автомат в некоторый момент вычисления, зависит только от префикса слова w, прочитанного к этому моменту. Эту зависимость мы и опишем формулами $\mathcal{LP}$-n-LTL.

Рассмотрим лежащий в основе автомата Маллера M неинициализированный конечный автомат $A = (\Sigma, Q, \Delta)$ и его инициализации $M(q, Q') = A(q, Q')$ для всевозможных состояний $q \in Q$ и подмножеств $Q' \subseteq Q$. Для упрощения обозначений мы будем использовать запись $M(q, Q')$ вместо $L(M(q, Q'))$.

Для каждого множества состояний $E_i \in \mathcal{F}$ построим следующие три формулы:

1. $\psi_1^i = \bigwedge_{q_e \in E_i} \mathbf{F}_{M(q_0, q_e)}\, \mathbf{true}$. Формула ψ_1^i выполняется на сверхслове w в том и только том случае, когда прогон автомата Маллера M на w проходит через все состояния множества E_i.

2. $\psi_2^i = \bigwedge_{q \in Q} \mathbf{G}_{M(q_0, q)} \left(\bigwedge_{q_e \in E_i} \mathbf{F}_{M(q, q_e) \setminus \{\varepsilon\}}\, \mathbf{true} \right)$. Данная формула выполняется на сверхслове w в том и только том случае, когда прогон автомата Маллера M на w проходит бесконечно часто через каждое состояние из множества E_i.

3. $\psi_3^i = \bigvee_{q \in Q} \mathbf{F}_{M(q_0, q)} \left(\mathbf{G}_{M(q, Q \setminus E_i)}\, \mathbf{false} \right)$. Эта формула выполняется на сверхслове w в том и только том случае, когда прогон автомата Маллера M на w проходит бесконечно часто только через состояния из множества E_i.

Из приведенных выше пояснений содержательного смысла формул ψ_1^i, ψ_2^i и ψ_3^i следует, что $\varphi_M = \bigvee_{i=1}^{m} (\psi_1^i \wedge \psi_2^i \wedge \psi_3^i)$ является искомой формулой. □

Таким образом, верна

Теорема 2. $\mathcal{LP}$-n-$LTL \equiv$ S1S.

7. Заключение

После опубликования статьи [5], в которой было проведено подробное изучение темпоральной логики LTL в свете ее применения в качестве средства описания поведения вычислительных систем, было предпринято несколько попыток расширить выразительные возможности этой логики. В статье [14] в синтаксис LTL были добавлены средства квантификации по базовым предикатам. Оказалось, что выразительность квантифицированного расширения $QLTL$ существенно превосходит

описательные возможности темпоральной логики LTL. Например, с использованием кванторов свойство $W_1^{2n} = \{w \in \Sigma^\omega \mid w(2i) = 1,\ i \in \mathbb{N}_0\}$ сверхслов в алфавите $\Sigma = \{0, 1\}$, невыразимое в логике LTL, описывается формулой ее расширения $QLTL$

$$\exists q(q \wedge \mathbf{G}(q \to \mathbf{X}\neg q) \wedge \mathbf{G}(\neg q \to q) \wedge \mathbf{G}(q \to p)) \ .$$

Автор статьи [19] предложил способ введения новых темпоральных операторов при помощи праволинейных грамматик. Слова, порождаемые этими грамматиками, задают шаблоны, на которых проверяется выполнимость формул в области действия темпорального оператора. Например, темпоральный оператор $\mathbf{E}p$, проверяющий указанное выше свойство W_1^{2n}, задается грамматикой из двух правил $V_0 \to pV_1$ и $V_1 \to \mathbf{true}\ V_0$. Подобным же образом шаблоны для описания семантики новых темпоральных операторов могут быть описаны при помощи конечных автоматов, как это показано в статьях [12, 16]. Не были обойдены вниманием и операторы неподвижной точки: логика линейного времени μ-LTL, обогащенная этими операторами, была введена и изучена в статье [18].

Для всех перечисленных расширений LTL (за исключением μ-LTL) было показано, что они имеют точно такие же выразительные способности, что и логика S1S, но при этом проблема выполнимости для всех этих логик остается PSPACE-полной, как и для LTL. Вводя в рассмотрение темпоральную логику $\mathcal{LP}$-LTL, мы не стремились всего лишь расширить выразительные возможности LTL как таковые. Нашей целью было создание адекватного языка для спецификации поведения реагирующих систем, моделируемых автоматами-преобразователями. Тем не менее, теорема 2 показывает, что выразительные возможности предложенного нами логического языка спецификаций совершили такой же скачок, как и другие расширения LTL, исследованные в статьях [12, 14, 16, 18, 19].

Идея параметризации темпоральных операторов не нова: почти такая же параметризация темпоральных операторов, что и в данной работе, была введена для динамического расширения логики LTL в статье [9]. По сути дела, фрагмент $\mathcal{LP}$-n-LTL имеет очень большое сходство с логикой $DLTL$, которая была введена в этой статье. Однако наше расширение LTL отличается, главным образом, тем, что базовые предикаты в нем также параметризованы. В дальнейшем мы собираемся провести сравнение выразительных возможностей логики $\mathcal{LP}$-n-LTL и динамических логик, включая $DLTL$ [9] и PDL [4].

Доказанная нами теорема 1 открывает еще одно направление исследований. Каждое из расширений логики LTL, предложенных в статьях [9, 14, 16, 18, 19], оказывается столь же выразительным, как одна из двух логик LTL и S1S. Однако, как показно в теореме 1, рассмотренный в нашей работе фрагмент $\mathcal{LP}$-1-LTL оказался более выразительным, нежели LTL, а как показано в теореме 2, другой фрагмент $\mathcal{LP}$-n-LTL оказался столь же выразительным, как и логика S1S. Мы высказываем предположение о том, что два рассмотренных в нашей работе фрагмента логики $\mathcal{LP}$-LTL имеют разные выразительные возможности. Если это предположение окажется справедливым, то фрагмент $\mathcal{LP}$-1-LTL будет являться естественным новым примером темпоральной логики, занимающей по выразительным возможностям промежуточное положение между хорошо известными логиками $(\mathbb{N}, <)$ и S1S. Мы полагаем, что эту гипотезу удастся доказать при помощи игр Эренфойхта–Фреше, которые были использованы в данной статье.

Список литературы / References

[1] Baier C., Katoen J., *Principles of Model Checking*, MIT Press, 2008.

[2] Clarke E. M., Gramberg O., Peled D. A., *Model Checking*, MIT Press, 1999.

[3] Etessami K., Wilke T., "An Until Hierarchy and Other Applications of an Ehrenfeucht-Fraisse Game for Temporal Logic", *Information and Computation*, **160**:1 (2000), 88–108.

[4] Fisher J. M., Ladner R. E., "Propositional dynamic logic of regular programs", *Journal of Computer and System Sciences*, **18** (1979), 194–211.

[5] Gabbay D., Pnueli A., Shelach S., Stavi J., "The temporal analysis of fairness", *Proceedings of 7-th ACM Symposium on Principles of Programming Languages*, 1980, 163-173.

[6] Гнатенко А. Р., Захаров В. А., "О сложности верификации конечных автоматов-преобразователей над коммутативными полугруппами", *Труды 18-й Международной конференции "Проблемы теоретической кибернетики"*, 2017, 68–70; [Gnatenko A. R., Zakharov V. A., "O slozhnosti verifikatsii avtomatov-preobrazovateley nad kommutativnymi polugruppami", *Trudy 18 Mezhdunarodnoj konferentsii "Problemy teoteticheskoy kibernetiki"*, 2017, 68–70, (in Russian).]

[7] Гнатенко А. Р., Захаров В. А., "О верификации конечных автоматов-преобразователей над полугруппами", *Труды Института системного программирования РАН*, **30**:3 (2018), 303–324; [Gnatenko A. R., Zakharov V. A., "On the model checking of finite state transducers over semigroups", *Proceedings of ISP RAS*, **30**:3, 303–324].

[8] Harel D., Kozen D., Parikh P., "Process logic: Expressiveness, decidability, completeness", *Journal of Computer and System Science*, **25**:2 (1982), 144–170.

[9] Henriksen J. J., Thiagarajan P. S., "Dynamic linear time temporal logic", *Annals of Pure and Applied Logic*, **96**:1–3 (1999), 187–207.

[10] Kamp J. A. W., *Tense Logic and the Theory of Linear Order, PhD thesis*, University of California, Los Angeles, 1968.

[11] Kozlova D. G., Zakharov V. A., "On the model checking of sequential reactive systems", *Proceedings of the 25th International Workshop on Concurrency, Specification and Programming (CS&P 2016)*, CEUR Workshop Proceedings, **1698**, 2016, 233–244.

[12] Kupferman O., Piterman N., Vardi M. Y., "Extended Temporal Logic Revisited", *Proceedings of 12-th International Conference on Concurrency Theory*, 2001, 519–535.

[13] Leucker M., Sanchez C., "Regular Linear Temporal Logic", *Proceedings of the 4-th International Colloquium on Theoretical Aspects of Computing*, 2007, 291–305.

[14] Manna Z., Wolper P., "Synthesis of Communicating Processes from Temporal Logic Specifications", *Lecture Notes in Computer Science*, **131**, 1981, 253–281.

[15] Thomas W., "Automata on infinite objects", *Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics*, 1990, 133–192.

[16] Vardi M. Y., Wolper P., "Yet Another Process Logic", *Lecture Notes in Computer Science*, **14**, 1983, 501–512.

[17] Vardi M. Y., Wolper P., "An Automata-Theoretic Approach to Automatic Program Verification", *Proceedings of the First Symposium on Logic in Computer Science*, 1986, 322–331.

[18] Vardi M. Y., "A Temporal Fixpoint Calculus", *Proceedings of the 15-th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1989, 250–259.

[19] Wolper P., "Temporal Logic Can Be More Expressive", *Information and Control*, **56**:1–2 (1983), 72–99.

[20] Wolper P., Boigelot B., "Verifying systems with infinite but regular state spaces", *Lecture Notes in Computer Science*, **1427**, 1998, 88–97.

[21] Zakharov V. A., "Equivalence checking problem for finite state transducers over semigroups", *Lecture Notes in Computer Science*, **9270**, 2015, 208–221.

[22] Захаров В. А., Темербекова Г. Г., "О минимизации конечных автоматов-преобразователей над полугруппами", *Моделирование и анализ информационных систем*, **23**:6 (2016), 741–753; English transl.: Zakharov V. A., Temerbekova G. G., "On the minimization of finite state trans-ducers over semigroups", *Automatic Control and Computer Sciences*, **51**:7 (2017), 523–530.

[23] Захаров В. А., Жайлауова Ш. Р., "О задаче минимизации последовательных программ", *Моделирование и анализ информационных систем*, **24**:4 (2017), 415–430; English transl.: Zakharov V. A., Jaylauova S. R., "On the minimization problem for sequential programs", *Automatic Control and Computer Sciences*, **51**:7 (2017), 689–700.

Gnatenko A. R., Zakharov V.A., "On the Expressive Power of Some Extensions of Linear Temporal Logic", *Modeling and Analysis of Information Systems*, **25**:5 (2018), 506–524.

DOI: 10.18255/1818-1015-2018-5-506-524

Abstract. One of the most simple models of computation which is suitable for representation of reactive systems behaviour is a finite state transducer which operates over an input alphabet of control signals and an output alphabet of basic actions. The behaviour of such a reactive system displays itself in the correspondence between flows of control signals and compositions of basic actions performed by the system. We believe that the behaviour of this kind requires more suitable and expressive means for formal specifications than the conventional LTL. In this paper, we define some new (as far as we know) extension $\mathcal{LP}$-LTL of Linear Temporal Logic specifically intended for describing the properties of transducers computations. In this extension the temporal operators are parameterized by sets of words (languages) which represent distinguished flows of control signals that impact on a reactive system. Basic predicates in our variant of the temporal logic are also languages in the alphabet of basic actions of a transducer; they represent the expected response of the transducer to the specified environmental influences. In our earlier papers, we considered a model checking problem for $\mathcal{LP}$-LTL and $\mathcal{LP}$-CTL and showed that this problem has effective solutions. The aim of this paper is to estimate the expressive power of $\mathcal{LP}$-LTL by comparing it with some well known logics widely used in the computer science for specification of reactive systems behaviour. We discovered that a restricted variant $\mathcal{LP}$-1-LTL of our logic is more expressive than LTL and another restricted variant $\mathcal{LP}$-n-LTL has the same expressive power as monadic second order logic S1S.

Keywords: temporal logics, expressive power, specification, verification, Buchi automata, infinite words

On the authors:
Anton R. Gnatenko, orcid.org/0000-0003-1499-2090, student,
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: gnatenko.cmc@gmail.com

Vladimir A. Zakharov, orcid.org/0000-0002-3794-9565, Dr. of Science, professor,
National Research University Higher School of Economics (HSE),
20 Myasnitskaya ulitsa, Moscow, 101000 Russia, e-mail: zakh@cs.msu.ru

Acknowledgments:
This work was supported by the Russian Foundation for Basic Research, Grant N 18-01-00854

Моделирование и анализ информационных систем. Т. 25, № 5 (2018), с. 525–533
Modeling and Analysis of Information Systems. Vol. 25, No 5 (2018), pp. 525–533

DOI: 10.18255/1818-1015-2018-5-525-533

УДК 517.9

О безопасности одно- и многоместных IFP-операторов

Дудаков С. М.

получена 10 сентября 2018

Аннотация. В работе изучается безопасность унарных операторов инфляционной неподвижной точки (IFP-операторов), то есть возможность их вычисления за конечное время. Такие операторы в точности соответствуют рекурсивным SQL-запросам, поэтому изучаемый вопрос имеет непосредственное отношение к базам данных. Исследуемая проблема возникает из-за того, что при одновременном применении в SQL запросе рекурсии и отношений универсума, например, сложения, может оказаться так, что процедура вычисления результата запроса зациклится. Более того, такая комбинация позволяет моделировать работу универсального вычислительного устройства, например, машины Тьюринга, поэтому вопрос о возможности вычисления SQL запроса за конечное время оказывается алгоритмически неразрешимым. В предыдущих работах были введены и изучены некоторые свойства универсумов, которые позволяют гарантировать возможность вычисления любых запросов за конечное время. Здесь мы изучаем вопрос о том, насколько существенна местность IFP-операторов в контексте их безопасности. Основным результатом настоящей работы является демонстрация того, что если ограничиться только унарными IFP-операторами, то не имеют места результаты, справедливые для IFP-операторов в общем случае без ограничения местности. Построен пример универсума, в котором все унарные IFP-операторы, не вложенные один в другой, безопасны. Вместе с тем в этом универсуме существуют небезопасные бинарные IFP-операторы, таким образом, при изменении местности безопасность может утрачиваться. Кроме того, существуют и небезопасные вложенные один в другой унарные операторы. Это контрастирует с общим случаем, в котором такое невозможно. Также существуют элементарно эквивалентные универсумы, в которых те же самые унарные IFP-операторы безопасными не являются. Такое поведение тоже отличается от поведения IFP-операторов произвольной местности.

Ключевые слова: инфляционная неподвижная точка, местность, безопасность

Для цитирования: Дудаков С. М., "О безопасности одно- и многоместных IFP-операторов", *Моделирование и анализ информационных систем*, **25**:5 (2018), 525–533.

Об авторах:
Дудаков Сергей Михайлович, orcid.org/0000-0003-2659-265X, доктор физ.-мат. наук, доцент,
Тверской государственный университет,
ул. Желябова, 33, г. Тверь, 170100 Россия, e-mail: sergeydudakov@yandex.ru

1. Введение

Традиция применения языков первого порядка для запросов к базам данных восходит к Кодду (см. [1]). Современные варианты языка SQL поддерживают как выразительные возможности языков первого порядка, так и некоторые расширения.

Например, рекурсивные запросы в точности соответствуют оператору инфляционной неподвижной точки (IFP-оператору, [5]). Также язык SQL позволяет применять в запросах функции и отношения универсума, из которого выбираются элементы базы данных. Таким образом, мы имеем конечную реляционную структуру (таблицы базы данных), вложенную в бесконечный универсум (см. [6]).

Одновременное использование IFP-операторов и отношений бесконечного универсума легко может привести к тому, что вычисление значения SQL-запроса зациклится. Легко можно показать, что даже обычной бесконечной функции следования (например, прибавления единицы для натуральных чисел) достаточно для моделирования любой машины Тьюринга с помощью IFP-операторов. Следовательно, зацикливание в таких случаях даже невозможно предсказать.

Возможность зацикливания принципиально отличает современный язык SQL от классической реляционной алгебры Кодда, в которой значение каждого запроса вычисляется за конечное число шагов. Последнее свойство мы будем называть *безопасностью*, а запросы, которые им обладают, — *безопасными*.

В работе [2] мы ввели класс универсумов, в которых любой запрос является безопасным, то есть зацикливание при их вычислении невозможно в принципе. Такие универсумы мы тоже называем *безопасными*.

В предыдущих наших исследованиях мы установили некоторые свойства, связанные с безопасностью универсумов. Например, в [4] найдены необходимые и достаточные условия безопасности универсума. Эти условия сводятся к некоторым свойствам первого порядка, следовательно, безопасными или небезопасными являются одновременно все элементарно эквивалентные универсумы. Поэтому можно говорить о безопасности даже не самих универсумов, а их элементарных теорий. В [3] безопасность произвольных вложенных IFP-операторов сведена к безопасности операторов, которые не вложены друг в друга, то есть применяются к формулам первого порядка.

Доказательства перечисленных в предыдущем абзаце утверждений включают в себя построение IFP-операторов произвольной местности. Однако на примере логики первого порядка хорошо известно, что многие алгоритмические свойства могут отличаться для унарного и многоместного случаев: скажем, логика унарных предикатов алгоритмически разрешима, а уже при наличии одного бинарного отношения теряет это свойство.

В настоящей работе мы исследуем некоторые свойства унарных IFP-операторов. Мы показываем, что многие свойства многоместных IFP-операторов на унарный случай не переносятся: безопасность унарных IFP-операторов без вложения не гарантирует безопасность ни вложенных унарных, ни бинарных IFP-операторов; существуют элементарно эквивалентные универсумы, в одном из которых все унарные IFP-операторы безопасны, а в другом — нет. Таким образом, местность IFP-операторов существенна при исследовании их свойств.

2. Определения

Мы рассматриваем расширение логики первого порядка (FO) с помощью оператора инфляционной неподвижной точки (IFP).

Определение 1 (см. [5]). *Формулы IFP-логики строятся как формулы первого порядка, а также с помощью* IFP-оператора*: если $\phi(\bar{x}, \bar{y})$ — формула в языке (сигнатуре) $\Sigma \cup \{Q^{(n)}\}$ со свободными переменными $\bar{x}$, $\bar{y}$, где Q — новый символ отношения, не входящий в Σ, то строка $\mathrm{IFP}_{Q(\bar{x})}(\phi)$ является формулой в языке Σ со свободными переменными $\bar{x}$, $\bar{y}$. Местность n символа отношения Q должна быть равна длине набора $\bar{x}$, она называется* местностью *IFP-оператора.*

Значения термов, атомных формул, булевых связок и кванторов определяются так же, как в FO-логике (см., например, [7]).

Определение 2 (см. [5]). *Пусть $\mathfrak{A}$ — алгебраическая система, $\phi(\bar{x}, \bar{y})$ — формула, семантика ϕ уже определена и ϕ содержит новый k-местный символ отношения Q. Пусть $\bar{a} \in \mathfrak{A}$ — это значения переменных $\bar{y}$. Тогда* значение формулы $\mathrm{IFP}_{Q(\bar{x})}(\phi)$ *определяется следующим способом. Рассмотрим последовательность множеств $Q_i \subseteq |\mathfrak{A}|^k$:*

$$Q_0^{\bar{a}} = \varnothing; \quad Q_{i+1}^{\bar{a}} = Q_i^{\bar{a}} \cup \{\bar{b} \in |\mathfrak{A}| : (\mathfrak{A}, Q_i^{\bar{a}}) \models \phi(\bar{b}, \bar{a})\},$$

для $i \in \omega$.

Допустим, $Q_n^{\bar{a}} = Q_{n+1}^{\bar{a}}$ для какого-то натурального n и $\bar{b}$ является значением переменных $\bar{x}$. Тогда формула $\mathrm{IFP}_{Q(\bar{x})}(\phi)(\bar{x}, \bar{a})$ истинна *при $\bar{b} \in Q_n^{\bar{a}}$ и* ложна *при $\bar{b} \notin Q_n^{\bar{a}}$. Если такого натурального n не существует, то значение формулы $\mathrm{IFP}_{Q(\bar{x})}(\phi)$ будем считать* неопределённым.

Если значение формулы $\mathrm{IFP}_{Q(\bar{x})}(\phi)(\bar{x}, \bar{a})$ определено для всех $\bar{a} \in \mathfrak{A}$, то назовём оператор $\mathrm{IFP}_{Q(\bar{x})}(\phi)$ безопасным *в $\mathfrak{A}$. Систему $\mathfrak{A}$ называем* безопасной*, если все IFP-операторы безопасны в $\mathfrak{A}$.*

Таким образом, мы считаем определёнными в точности те IFP-операторы, значение которых может быть вычислено за конечное число шагов.

Нетрудно видеть, что для каждого натурального i формула $Q_i^{\bar{a}}(\bar{x})$ эквивалентна следующей формуле $\psi_i(\bar{x}, \bar{a})$:

$$\psi_0 \equiv [\mathrm{false}]; \quad \psi_{i+1} \equiv \left[\psi_i \vee (\phi)^{Q(\bar{t})}_{\psi_i(\bar{t},\bar{a})}\right]. \tag{1}$$

Здесь формула $(\phi)^{Q(\bar{t})}_{\psi_i(\bar{t},\bar{a})}$ получена из ϕ заменой каждой атомной подформулы вида $Q(\bar{t})$ на $\psi_i(\bar{t}, \bar{a})$.

3. Теория T и ее свойства

Рассмотрим сигнатуру, которая содержит два унарных функциональных символа: s и d. По смыслу это будут функции следования и предшествования соответственно. Точнее, теория задаётся следующими аксиомами:

$$(\forall x)(s(d(x)) = x \wedge d(s(x)) = x),$$

а также следующими двумя аксиомами для всех натуральных $n > 0$:

$$(\exists x) \underbrace{\left(s^n(x) = x \wedge \bigwedge_{0<i<n} s^i(x) \neq x\right)}_{\phi_n(x)};$$

$$(\forall x)(\forall y)\Big(\phi_n(x) \wedge \phi_n(y) \to \bigvee_{0\le i<n} y = s^i(x)\Big).$$

С помощью $s^i(x)$ обозначен терм

$$\underbrace{s(s(\cdots s(}_{i \text{ раз}} x)\cdots)),$$

в частности $s^0(x) = x$. Из этих аксиом непосредственно вытекает, что каждая модель теории T для каждого натурального $n > 0$ содержит единственный «цикл» длины n, образованный функцией s в одном направлении и функцией d в противоположном. Для краткости будем называть этот цикл «n-циклом». Формула $\phi_n(x)$ говорит, что x принадлежит n-циклу. Следовательно, в теории T определимы одноместные предикаты R_n, означающие принадлежность n-циклам: $R_n(x) \equiv \phi_n(x)$.

Для удобства мы будем записывать $u + n$ и $u - n$ вместо $s^n(u)$ и $d^n(u)$ соответственно. Тогда будет справедливо следующее равенство: $(u + n) + m = u + (n + m)$ для любых целых n и m.

Теорема 1. *Теория T допускает элиминацию кванторов.*

Доказательство. Как обычно (см., например, [7]), можно ограничиться элиминацией одного квантора существования добавленного к элементарной конъюнкции, то есть из формулы вида $(\exists u)\theta$, где θ — элементарная конъюнкция.

Прежде всего, отметим, что отношения R_n определяются бескванторными формулами ϕ_n. Поэтому мы можем без ограничения общности выполнять элиминацию кванторов в сигнатуре, обогащённой символами R_n.

Нетрудно видеть, что равенство $u + k = t$ эквивалентно $u = t - k$, а формула $R_m(u + k)$ эквивалентна $R_m(u)$ для любых целых k и натуральных $m > 0$. Поэтому мы можем полагать, что все атомные формулы в θ имеют вид $u = u + n$, $u = t$ или $R_n(u)$, где t не содержит u.

Равенство $u = u + n$ эквивалентно дизъюнкции

$$\bigvee_{d|n} R_d(u),$$

в которой $d|n$ означает делимость. Следовательно, все равенства $u = u + n$ могут быть заменены дизъюнкцией формул вида $R_d(u)$.

Теперь мы можем считать, что формула $(\exists u)\theta$ имеет такой вид:

$$(\exists u)\theta \equiv (\exists u)\Big(\bigwedge_j R_{n_j}(u) \wedge \bigwedge_j \neg R_{m_j}(u) \wedge \bigwedge_k u = t_k \wedge \bigwedge_\ell u \ne r_\ell\Big), \tag{2}$$

где все термы t_k и r_ℓ не содержат u.

Если в конъюнкции (2) есть хоть одно равенство вида $u = t_k$, то квантор удаляется обычной заменой u на t_k: $(\exists u)\theta \equiv (\theta)^u_{t_k}$. Поэтому в дальнейшем мы рассматриваем только случай, когда в формуле (2) равенств $u = t_k$ нет.

Все формулы вида $R_{n_j}(u)$ для различных n_j попарно несовместны. Отсюда можно сделать вывод, что в (2) формула $R_n(u)$ должна быть единственной или вовсе отсутствовать. По той же причине, если в (2) присутствует $R_n(u)$, то из неё следуют все $\neg R_{m_j}(u)$ для $m_j \ne n$. Исходя из вышесказанного можно сделать вывод, что для формулы (2) имеет место один из трёх случаев:

1) формула (2) содержит одну формулу $R_n(u)$ и не содержит никаких $\neg R_{m_j}(u)$;

2) формула (2) не содержит $R_n(u)$, но содержит какие-то $\neg R_{m_j}(u)$;

3) формула (2) не содержит ни того, ни другого.

В случае 2 мы имеем формулу

$$(\exists u)\theta \equiv (\exists u)\Big(\bigwedge_j \neg R_{m_j}(u) \wedge \bigwedge_\ell u \neq r_\ell\Big). \tag{3}$$

Пусть значение каждого терма r_ℓ принадлежит соответствующему k_ℓ-циклу. Так как существуют n-циклы для всех положительных n, то мы можем выбрать n-цикл так, что $n \neq m_j$ и $n \neq k_\ell$ для всех j и ℓ. Поскольку каждое u из этого n-цикла удовлетворяет формулам $\neg R_{m_j}(u)$ и $u \neq r_\ell$, то формула (3) истинна.

Случай 3 рассматривается точно так же.

Рассмотрим теперь первый случай, когда формула имеет вид

$$(\exists u)\theta \equiv (\exists u)\Big(R_n(u) \wedge \bigwedge_\ell u \neq r_\ell\Big). \tag{4}$$

Добавим к (4) тождественно истинные дизъюнкции $(R_n(r_\ell) \vee \neg R_n(r_\ell))$ для всех ℓ, раскроем скобки и построим дизъюнктивную нормальную форму. В результате получим дизъюнкцию формул вида

$$(\exists u)\Big(R_n(u) \wedge \bigwedge_\ell u \neq r_\ell\Big) \wedge \bigwedge_\ell R_n^*(r_\ell),$$

в которых $R_n^*(r_\ell)$ означает $R_n(r_\ell)$ или $\neg R_n(r_\ell)$. Очевидно, что из $R_n(u)$ и $\neg R_n(r_\ell)$ автоматически следует $u \neq r_\ell$. Значит, мы можем исключить неравенство $u \neq r_\ell$ при наличии $\neg R_n(r_\ell)$. Поэтому получаем

$$(\exists u)\Big(R_n(u) \wedge \bigwedge_{\ell'} u \neq r_{\ell'}\Big) \wedge \bigwedge_{\ell'} R_n(r_{\ell'}) \wedge \bigwedge_{\ell''} \neg R_n(r_{\ell''}), \tag{5}$$

причём $\ell' \neq \ell''$ для всех ℓ' и ℓ''.

Если в формуле (5) отсутствуют неравенства, то она эквивалентна следующей бескванторной формуле:

$$\bigwedge_{\ell'} R_n(r_{\ell'}) \wedge \bigwedge_{\ell''} \neg R_n(r_{\ell''}),$$

так как $(\exists u)R_n(u)$ истинно.

В противоположном случае формула (5) содержит хотя бы одно неравенство: $u \neq r_1$. Тогда формула (5) истинна в том и только том случае, когда n-цикл содержит элементы, кроме $r_{\ell'}$, для всех ℓ'. Следовательно, (5) эквивалентно

$$\Big(\bigvee_{i=1}^{n-1} \bigwedge_{\ell'} \Big(R_n(r_{\ell'}) \wedge r_1 + i \neq r_{\ell'}\Big)\Big) \wedge \bigwedge_{\ell''} \neg R_n(r_{\ell''}). \tag{6}$$

При $n = 1$ пустая дизъюнкция, как обычно, считается ложной.

Итак, мы рассмотрели все возможные варианты формулы (2) и показали, как удалить квантор в каждом из них. Следовательно, теория T допускает элиминацию кванторов. □

Так как сигнатура не содержит констант или пропозициональных символов, то сразу получаем

Следствие 1. *Теория T полна.*

Введём ещё одно техническое определение.

Пусть формула ϕ содержит атомные формулы вида $R_{n_u}(u)$ и $v = u + k_{u,v}$ для каких-то целых n_u и $k_{u,v}$, $\bar{y}$ — какой-то фиксированный набор свободных в ϕ переменных. Тогда *$\bar{y}$-вес формулы* ϕ (обозначаем его с помощью $w_{\bar{y}}(\phi)$) — это наибольшее из всех таких n_u и $k_{u,v}$, что ни u, ни v не содержатся в $\bar{y}$.

Следствие 2. *В доказательстве теоремы 1 имеет место $w_{\bar{y}}(\theta') \leq 3w_{\bar{y}}(\theta)$, где θ' — это результат элиминации квантора из формулы $(\exists x)\theta$.*

Доказательство. Вес возрастает только при подстановке $(\theta)^u_{t_k}$, но не более чем в два раза, и при построении (6), но не более чем в три раза. □

Индукцией по количеству кванторов легко получаем

Следствие 3. *Если формула ψ имеет q кванторов, то эквивалентная ψ бескванторная формула ψ' имеет вес $w_{\bar{y}}(\psi') \leq 3^q w_{\bar{y}}(\psi)$.*

4. IFP-операторы в моделях теории T

Атомная модель $\mathfrak{A}_0$ теории T содержит по одному n-циклу для каждого $n > 0$ и не содержит никаких других элементов.

Теорема 2. *Для каждой FO-формулы $\phi(x, \bar{y})$ (с новым унарным предикатом Q) унарный оператор $\mathrm{IFP}_{Q(x)}(\phi)$ безопасен в $\mathfrak{A}_0$.*

Доказательство. Допустим, что формула ϕ содержит q кванторов, и в наборе $\bar{a}$ значений $\bar{y}$ каждое a_j принадлежит соответствующему m_j-циклу. Выберем натуральное w такое, что $m_j \leq w$ для всех j и $w_{\bar{y}}(\phi) \leq w$.

Рассмотрим последовательность формул $\psi_i(x, \bar{y})$, определённую в (1). Индукцией по i докажем, что каждая формула $\psi_i(x, \bar{y})$ эквивалентна бескванторной формуле $\bar{y}$-веса не более $3^{2q}w$. Для $i = 1$ это непосредственно получается из теоремы 1 и следствия 3.

Предположим теперь, что формула $\psi_i(x, \bar{y})$ является булевой комбинацией атомных формул вида $R_m(x)$ и $x = y_j + k_j$, где $m \leq 3^q w$. Когда мы выполняем подстановку формулы $\psi_i(u+\ell, \bar{y})$ вместо $Q(u+\ell)$ в формуле ϕ, мы получаем новые подформулы вида $R_m(u + \ell)$ и $u + \ell = y_j + k_j$, что эквивалентно $R_m(u)$ и $u = y_j + (k_j - \ell) = y_j + k'_j$ соответственно. Присутствие всех этих новых подформул не может увеличить вес формулы в процедуре элиминации квантора. В самом деле, мы не получим новых подформул типа $u = u + n$ и, следовательно, новых подформул типа $R_d(u)$, где u не входит в $\bar{y}$. Мы не можем увеличить $\bar{y}$-вес при подстановке $(\theta)^u_{y_j+k'_j}$ и при использовании термов $y_j + k'_j$ в (6). Следовательно, после элиминации квантора ψ_{i+1} имеет вес не более $3^{2q}w$.

Итак, каждая формула $\psi_i(x, \bar{y})$ является булевой комбинацией атомных формул вида $R_m(x)$ и $x = y_j + k_j$, где $m \leq 3^q w$. Поскольку $y_j + m_j = y_j$, то можно полагать, что $0 \leq k_j < m_j$ для всех j. Но всего существует ограниченно много попарно неэквивалентных формул такого вида. Следовательно, $\psi_i \equiv \psi_{i+1}$ для некоторого i и оператор $\mathrm{IFP}_{Q(x)}(\phi)$ безопасен. □

Предыдущая теорема неверна для бинарных IFP-операторов.

Теорема 3. *Бинарный оператор*

$$\mathrm{IFP}_{Q(x,y)}(x = y \vee Q(x, d(y)))$$

небезопасен в $\mathfrak{A}_0$.

Доказательство. Множество Q_i содержит все пары вида $(a, a + i)$, $i = 0, \ldots, i - 1$. Если a принадлежит $n+1$-циклу, то $(a, a+i) \in Q_{i+1} \setminus Q_i$. Следовательно, последовательность множеств Q_i неограниченно возрастает при увеличении i, а IFP-оператор небезопасен. □

Безопасными могут не быть и вложенные унарные IFP-операторы.

Теорема 4. *Существуют вложенные унарные IFP-операторы, внешний из которых небезопасен в* $\mathfrak{A}_0$.

Доказательство. Рассмотрим формулу $\psi(y, x)$:

$$\mathrm{IFP}_{Q(x)}(x = y \vee Q(d(x))).$$

Если значение y равно a, то Q_i^a будет содержать $a, a + 1, \ldots, a + (i - 1)$. Если a принадлежит n-циклу, то $Q_n^a = Q_{n+1}^a$, поэтому $\psi(y, x)$ истинно тогда и только тогда, когда x и y принадлежат одному циклу.

Рассмотрим теперь формулу $\phi(y, z, x)$:

$$\mathrm{IFP}_{Q(x)}(x = y \vee x = z \vee Q(d(x)) \wedge \\ \wedge (\exists x_1, x_2)(x_1 \neq x_2 \wedge \neg Q(x_1) \wedge Q(d(x_1)) \wedge \neg Q(x_2) \wedge Q(d(x_2)))).$$

Пусть a и b являются значениями переменных y и z соответственно. Тогда на первом шаге построения мы добавляем a и b к $Q_1^{a,b}$. Далее на каждом шаге к $Q_i^{a,b}$ добавляются по два элемента, следующих за уже имеющимися. Такой процесс завершится, когда один из двух следующих элементов уже будет содержаться в $Q_i^{a,b}$, иными словами, все элементы соответствующего цикла попали в $Q_i^{a,b}$. Следовательно, формула $\phi(y, z, x)$ означает, что для какого-то m все $y + i$ и $z + j$ попарно различны, $i, j = 0, \ldots, m$, и $x = y + i$ или $x = z + j$ для каких-то $i, j = 0, \ldots, m$. Если a и b принадлежат различным циклам, то предшественник a или b (в зависимости от того, какой из них длиннее) не попадёт в $Q^{a,b}$.

Таким образом, формула

$$\theta(y, z) \equiv \neg\psi(y, z) \wedge [y \neq z \wedge \phi(y, z, d(y))]$$

означает, что цикл, в котором лежит y, короче, чем тот, в котором лежит z. Поэтому мы имеем предпорядок θ, фактор-порядок которого будет изоморфен множеству

натуральных чисел. Следовательно, следующая функция D будет функцией предшествования для циклов:

$$[D(z) = y] \equiv [\theta(y, z) \wedge \neg(\exists u)(\theta(y, u) \wedge \theta(u, z))].$$

Поэтому оператор $\mathrm{IFP}_{P(x)}(x = s(x) \vee P(D(x)))$ небезопасен. □

Теория T имеет неатомные модели $\mathfrak{B}$, которые, кроме циклов, содержат «линии» бесконечные в обе стороны. В таких моделях унарные IFP-операторы тоже могут оказаться небезопасными.

Теорема 5. *Унарный оператор* $\mathrm{IFP}_{Q(x)}(x = y \vee Q(d(x)))$ *небезопасен в любой неатомной модели* $\mathfrak{B}$ *теории* T.

Доказательство. Если значение a переменной y принадлежит бесконечной «линии», то Q_i^a содержит $a + j$ для $j = 0, \ldots, i - 1$. Следовательно, не существует натурального i такого, что $Q_i^a = Q_{i+1}^a$. □

5. Заключение

Мы показали, что свойства унарных IFP-операторов отличаются от свойств IFP-операторов произвольной местности: все невложенные унарные IFP-операторы могут быть безопасными, а вложенные и бинарные — нет, безопасность совокупности всех унарных невложенных операторов может быть различной в элементарно эквивалентных универсумах.

В связи с этим возникают следующие вопросы:

- Можно ли аналогичным образом построить пример универсума, в котором различались свойства безопасности бинарных и тернарных IFP-операторов?
- Можно ли построить полную теорию T, во всех моделях которой унарные невложенные IFP-операторы были бы безопасны, а бинарные — нет?

Список литературы / References

[1] Codd E. F., “Relational completeness of data base sublanguages”, *Database Systems*, ed. Rustin R., Prentice-Hall, 1972, 33–64.

[2] Дудаков С. М., “О безопасности рекурсивных запросов”, *Вестник ТвГУ. Серия: Прикладная математика*, 2012, № 4, 71–80; [Dudakov S. M., “On safety of recursive queries”, *Vestnik TvGU. Ser.: Prikl. Matem. [Herald of Tver State University. Ser.: Appl. Math.]*, 2012, № 4, 71–80, (in Russian).]

[3] Дудаков С. М., “О безопасности IFP-операторов и рекурсивных запросов”, *Вестник ТвГУ. Серия: Прикладная математика*, 2013, № 2, 5–13; [Dudakov S. M., “On safety of IFP-operators and recursive queries”, *Vestnik TvGU. Ser.: Prikl. Matem. [Herald of Tver State University. Ser.: Appl. Math.]*, 2013, № 2, 5–13, (in Russian).]

[4] Dudakov S. M., “On inflationary fix-point operators safety”, *Lobachevskii J. Math.*, **36**:4 (2015), 328–331.

[5] Gurevich Y., Shelah S., “Fixed-point extensions of first-order logic”, *Annals of Pure and Applied Logic*, **32** (1986), 265–280.

[6] Kanellakis P., Kuper G., Revesz P., “Constraint query languages”, *J. of Computer and System Sciences*, **51**:1 (1995), 26–52.

[7] Marker D., *Model theory: an introduction*, Springer-Verlag, New York, 2002.

Dudakov S. M., "On Safety of Unary and Non-unary IFP-operators", *Modeling and Analysis of Information Systems*, **25**:5 (2018), 525–533.

DOI: 10.18255/1818-1015-2018-5-525-533

Abstract. In this paper, we investigate the safety of unary inflationary fixed point operators (IFP-operators). The safety is a computability in finitely many steps. IFP-operators exactly correspond to recursive SQL-queries hence this problem has a value for database theory. The problem appears from the fact that if recursive queries contain universe functions and relations, then its execution can fall into an infinite loop. Moreover, universal computational devices (Turing machines et al.) can be modelled by such queries. Hence the problem of the finite computability for such queries is undecidable. In our previous works we established some properties of a universe which imply the finite computability of all IFP-operators in the universe. Here, we investigate a connection between an arity of IFP-operators and their safety. We prove that some results for general IFP-operators don't hold for unary ones. We construct a universe where all unary unnesed IFP-operators are safe. But in this universe there exist unsafe nested unary IFP-operators and unsafe unnested binary IFP-operators. This differs from general IFP-operators because in general case the safety of all unnesed IFP-operators implies the safety of all IFP-operators. Also there exist elementary equivalent universes where some unary unnesed IFP-operators become unsafe. For general IFP-operators it is also impossible.

Keywords: inflationary fixed point, arity, safety

On the authors:
Sergey M. Dudakov, orcid.org/0000-0003-2659-265X, Dr. of Science,
Tver State University,
33 Zhelyabova str., Tver 170100, Russia, e-mail: sergeydudakov@yandex.ru

Моделирование и анализ информационных систем. Т. 25, № 5 (2018), с. 534–548
Modeling and Analysis of Information Systems. Vol. 25, No 5 (2018), pp. 534–548

Синтез и преобразования программ
Program Synthesis and Transformations

DOI: 10.18255/1818-1015-2018-5-534-548

UDC 519.681.3

Polyprograms and Polyprogram Bisimulation

Grechanik S. A.

Received 10 September 2018

Abstract. A *polyprogram* is a generalization of a program which admits multiple definitions of a single function. Such objects arise in different transformation systems, such as the Burstall–Darlington framework or equality saturation. In this paper, we introduce the notion of a polyprogram in a non-strict first-order functional language.

We define denotational semantics for polyprograms and describe some possible transformations of polyprograms, namely we present several main transformations in two different styles: in the style of the Burstall–Darlington framework and in the style of equality saturation. Transformations in the style of equality saturation are performed on polyprograms in decomposed form, where the difference between functions and expressions is blurred, and so is the difference between substitution and unfolding. Decomposed polyprograms are well suited for implementation and reasoning, although they are not very human-readable.

We also introduce the notion of *polyprogram bisimulation* which enables a powerful transformation called *merging by bisimulation*, corresponding to proving equivalence of functions by induction or coinduction. Polyprogram bisimulation is a concept inspired by bisimulation of labelled transition systems, but yet it is quite different, because polyprogram bisimulation treats every definition as self-sufficient, that is a function is considered to be defined by any of its definitions, whereas in an LTS the behaviour of a state is defined by all transitions from this state.

We present an algorithm for enumerating polyprogram bisimulations of a certain form. The algorithm consists of two phases: enumerating prebisimulations and converting them to proper bisimulations. This separation is required because polyprogram bisimulations take into account the possibility of parameter permutation. We prove correctness of this algorithm and formulate a certain weak form of its completeness.

The article is published in the author's wording.

Keywords: polyprograms, program transformation, equality saturation, bisimulation

For citation: Grechanik S. A., "Polyprograms and Polyprogram Bisimulation", *Modeling and Analysis of Information Systems*, **25**:5 (2018), 534–548.

On the authors:
Sergei A. Grechanik, orcid.org/0000-0001-8575-9689, PhD,
Keldysh Institute of Applied Mathematics,
4 Miusskaya sq., Moscow 125047, Russia, e-mail: sergei.grechanik@gmail.com

Acknowledgments:
This work was supported by Russian Foundation for Basic Research, grant No. 18-31-00412.

Introduction

Many program transformation methods can be seen as special cases of the Burstall-Darlington framework [3]. The idea behind this framework consists in viewing a program as a set of equations and then transforming this set by inferring new equations. Such a set of equations is essentially a program without the uniqueness constraint on the definitions of its functions (i.e. each function may have several definitions), thus we propose to call it a *polyprogram* (short for "polyvariant program", a term coined by Mikhail Bulyonkov).

Equality saturation [8] may also be considered an instance of the Burstall-Darlington framework if we restrict ourselves to so-called *decomposed polyprograms*. In decomposed polyprograms every definition has a very simple form containing only one nontrivial language construct, thus decomposed polyprograms are essentially closer to ASTs and E-PEGs (E-PEG is a Program Expression Graph with an equivalence relation on nodes [8]), definitions of polyprograms corresponding to nodes and outgoing edges of E-PEGs, and functions corresponding to classes of node equivalence. Decomposed polyprograms can be represented as graphs, or, more precisely, directed hypergraphs, whose nodes correspond to functions and hyperedges to definitions. Thus decomposed polyprograms are better for implementation and formulation of transformations. Every complex definition can be split into several simple definitions by introducing intermediate functions, so every polyprogram can be transformed into a decomposed one.

One of the transformation rules of the Burstall-Darlington framework is called *redefinition*. It allows replacing one function for another if they have isomorphic recursive definitions. In this paper we show how this rule can be formulated using the notion of *polyprogram bisimulation*, which has the benefit of dealing with situations when the definitions are not exactly isomorphic (e.g. the functions are equal only up to argument permutation). Thus, we prefer to call this transformation rule *merging by bisimulation*.

This paper is a continuation of the work on equality saturation for functional languages [5]. In that previous paper the theory behind the implementation wasn't described thoroughly enough, in particular, the semantics wasn't discussed at all, the notion of polyprogram bisimulation wasn't presented, and thus there was no proof of correctness of the bisimulation enumeration algorithm. The present paper discusses these topics in more detail, and its contributions are as follows:

- Articulation of the notion of a polyprogram.
- A polyprogram-based formulation of equality saturation which shows the connection between equality saturation and the Burstall-Darlington framework.
- The notion of polyprogram bisimulation and an algorithm for enumerating polyprogram bisimulations together with a proof of its correctness. This is the main contribution, and most of the paper is devoted to this topic.

The paper is structured as follows: first of all, we describe the language we use throughout the paper and give the definitions of a polyprogram and a decomposed polyprogram in this language (Section 1.), then we show some basic transformation rules (Section 2.), and after that we introduce the notion of polyprogram bisimulation and present an algorithm for enumerating bisimulations (Section 3.).

1. Polyprograms

In this paper we use a simple first-order language. We denote variables with letters x, x_i (from $\mathcal{X}$), functions names with f, f_i (from a set of functional symbols $\mathcal{F}$), and constructors with C, C_i. A set of functional symbols is just a set F equipped with an arity function $arity : F \to \mathbb{N}$.

Definition 1. *A* polyprogram *(in our language) is a set of definitions of the form* $f(x_1, \ldots, x_n) \equiv e$ *where* e *has the following form*[1]*:*

$$e ::= x \mid f(e_1, \ldots, e_m) \mid C(e_1, \ldots, e_m) \mid \textbf{case } e_0 \textbf{ of } \{ \overline{C_i(\overline{x_{ij}}) \ \to \ e_i;} \}$$

where there is no variable duplication in case patterns and in left hand sides of definitions.

In a polyprogram each function is allowed to have any number of definitions. The intention is that such definitions should be semantically equal, but may be different performance-wise. Here is an example of a polyprogram:

$$\begin{aligned} \mathrm{not}(t) &\equiv \textbf{case } t \textbf{ of } \{F \to T; T \to F\} \\ \mathrm{even}(x) &\equiv \textbf{case } x \textbf{ of } \{Z \to T; S(y) \to \mathrm{odd}(y)\} \\ \mathrm{even}(x) &\equiv \mathrm{not}(\mathrm{odd}(x)) \\ \mathrm{odd}(x) &\equiv \textbf{case } x \textbf{ of } \{Z \to F; S(y) \to \mathrm{even}(y)\} \\ \mathrm{odd}(x) &\equiv \mathrm{not}(\mathrm{even}(x)) \end{aligned}$$

Note that the functions even and odd have two definitions each.

Definition 2. *A polyprogram in decomposed form, or just* decomposed polyprogram*, is a polyprogram such that all right hand sides of its definitions have the following form:*

$$\begin{aligned} &e ::= r \mid x \mid f(r_1, \ldots, r_m) \mid C(r_1, \ldots, r_m) \mid \textbf{case } r_0 \textbf{ of } \{ \overline{C_i(\overline{x_{ij}}) \ \to \ r_i;} \} \\ &r ::= f(x_1, \ldots, x_l), \textit{ where all } x_j \textit{ differ from each other} \end{aligned}$$

We call expressions of the form $f(x_1, \ldots, x_l)$, where variables are different, *elementary calls*. That is, every right hand side of a decomposed polyprogram is either an elementary call or an expression such that each of its maximal proper subexpressions is an elementary call.

Decomposed form is not very human-readable, but it is better suited for reasoning and implementation. Consider the following polyprogram consisting of one definition:

$$f(x, z) \equiv \textbf{case } x \textbf{ of } \{Z \to Z; S(y) \to f(y, y)\}$$

To transform it into a decomposed polyprogram, we have to factor out subexpressions x, Z, y, and $f(y, y)$ (the last one because it has a duplicated variable). This gives us the following decomposed polyprogram:

$$\begin{aligned} &f(x, z) \equiv \textbf{case } id(x) \textbf{ of } \{Z \to g(); S(y) \to h(y)\} \\ &id(x) \equiv x \\ &g() \equiv Z \\ &h(y) \equiv f(id(y), id(y)) \end{aligned}$$

[1] $\overline{E\langle e_i \rangle}$ expands to $E\langle e_1 \rangle, \ldots, E\langle e_n \rangle$ for some $n = \max i$

1.1. Polyprogram semantics

It is straightforward to define denotational semantics for polyprograms. However, instead of considering only the least fixed point, we consider all fixed points, or *models*, because this makes semantics compositional, i.e. we can replace polyprogram fragments with semantically equivalent fragments without changing the meaning of the whole polyprogram. This property would not hold if we considered only the least fixed point, because the equivalence based on the least fixed point semantics is too coarse.

Our polyprograms operate first-order values from the set A which is the greatest solution of the following equation (i.e. it includes both finite and infinite data built out of constructors):

$$A = \{C(a_1, \ldots, a_n) \mid a_i \in A,\ C \text{ is a constructor}\} \cup \{\bot\}.$$

Let D be the set of continuous functions [10] over A of arbitrary arity, i.e. $D = \bigcup_n [A^n \to A]$. Now let's define the notion of an interpretation.

Definition 3. *Let P be a polyprogram with the set of function names F. Then an* interpretation *of P is a function $\eta : F \to D$ such that $arity(\eta(f)) = arity(f)$.*

Now let's define the valuation of a term t given an interpretation η and a valuation of variables $\nu : \mathcal{X} \to A$, written $[\![t]\!]_{\eta,\nu}$:

$$\begin{aligned}
&[\![x]\!]_{\eta,\nu} = \nu(x)\\
&[\![f(e_1, \ldots, e_n)]\!]_{\eta,\nu} = \eta(f)([\![e_1]\!]_{\eta,\nu}, \ldots, [\![e_n]\!]_{\eta,\nu})\\
&[\![C(e_1, \ldots, e_n)]\!]_{\eta,\nu} = C([\![e_1]\!]_{\eta,\nu}, \ldots, [\![e_n]\!]_{\eta,\nu})\\
&[\![\textbf{case } e_0 \textbf{ of} \{\overline{C_i(y_1, \ldots, y_m) \to e_i}\}]\!]_{\eta,\nu} = [\![e_k]\!]_{\eta,\nu\{y_i \to a_i\}}\\
&\qquad \text{where } [\![e_0]\!]_{\eta,\nu} = C_k(a_1, \ldots, a_m) \text{ for some } k \in \{1, \ldots, \max i\}
\end{aligned}$$

Given a definition d, its valuation $[\![d]\!]_\eta$ is a Boolean value defined as follows:

$$[\![e_1 \equiv e_2]\!]_\eta = (\forall \nu. [\![e_1]\!]_{\eta,\nu} = [\![e_2]\!]_{\eta,\nu})$$

Definition 4. *An interpretation μ is called a* model *of a polyprogram P if for every definition $d \in P$, $[\![d]\!]_\mu$ is true.*

2. Polyprogram transformation rules

According to both the Burstall-Darlington framework and equality saturation, polyprograms should be transformed with some rules which add new function definitions to a polyprogram. After that a new program may be extracted from the polyprogram by choosing a single definition for each function.

We write transformation rules as $P_1 \mapsto P_2$. Application of such a rule to a polyprogram consists in replacing the subpolyprogram corresponding to the left hand side up to function and variable renaming with the right hand side. We assume that rules may add new definitions and functions, and also remove some definitions

We will consider only a couple of basic rules in two different styles: in the style of the Burstall-Darlington framework and in the style of equality saturation. The latter one assumes that polyprograms are in decomposed form.

The main difference between the two styles is that rules of equality saturation are more local and fine-grained, because equality saturation was originally designed to perform mostly intraprocedural optimizations (inside function bodies) by rewriting expressions. However, for decomposed polyprograms the difference between expressions and functions becomes blurred, so interprocedural transformations become as naturally expressible in equality saturation as intraprocedural.

2.1. Rules in the style of the Burstall-Darlington framework

Unfolding

$$\left\{\begin{array}{l} f(\overline{x_i}) \equiv E\langle g(\overline{e_j})\rangle \\ g(\overline{y_j}) \equiv H \end{array}\right\} \mapsto \left\{\begin{array}{l} f(\overline{x_i}) \equiv E\langle g(\overline{e_j})\rangle \\ g(\overline{y_j}) \equiv H \\ f(\overline{x_i}) \equiv E\langle H\{\overline{y_j \mapsto e_j}\}\rangle \end{array}\right\}$$

This rule substitutes a function call with the function body.

Folding

$$\left\{\begin{array}{l} f(\overline{x_i}) \equiv H\langle E\{\overline{y_j \mapsto z_j}\}\rangle \\ g(\overline{y_j}) \equiv E \end{array}\right\} \mapsto \left\{\begin{array}{l} f(\overline{x_i}) \equiv H\langle E\{\overline{y_j \mapsto z_j}\}\rangle \\ g(\overline{y_j}) \equiv E \\ f(\overline{x_i}) \equiv H\langle g(\overline{z_j})\rangle \end{array}\right\}$$

This rule does the inverse: it replaces an instance of some function's body with a call of this function. Note that it is less powerful than the corresponding rule from the paper by Burstall and Darlington [3] because it allows replacing only renamings of g's body, not arbitrary special cases.

2.2. Rules in the style of equality saturation

Decomposed polyprograms are very similar to E-PEGs from the work of Tate et al. on equality saturation [8]. They enable more effective sharing of subexpressions, which is crucial for big polyprograms, especially when a simple heuristic-free rule application strategy is used, as in equality saturation.

However, rules in the form from the previous subsection cannot be applied to decomposed polyprograms because their left hand sides are not in decomposed form. One of the solutions to this problem is to rewrite the rules in such a way that both their sides are in decomposed form. In this case the rules will not only be applicable to decomposed polyprograms but also will preserve them in decomposed form.

Transitivity with symmetry

$$\left\{\begin{array}{l} f(\overline{x_i}) \equiv E \\ g(\overline{y_j}) \equiv E \end{array}\right\} \mapsto \left\{\begin{array}{l} f(\overline{x_i}) \equiv E \\ g(\overline{y_j}) \equiv E \\ f(\overline{x_i}) \equiv g(\overline{y_j}) \end{array}\right\}$$

This rule is analogous to folding. It infers function equivalence from their having coinciding definitions. The correctness of this rule follows from the transitivity and symmetry of equality, hence the name.

Congruence

$$\left\{\begin{array}{l} g(\overline{y_j}) \equiv h(\overline{y_{\theta(j)}}) \\ D\langle g(\overline{e_j})\rangle \end{array}\right\} \mapsto \left\{\begin{array}{l} g(\overline{y_j}) \equiv h(\overline{y_{\theta(j)}}) \\ D\langle h(\overline{e_{\theta(j)}})\rangle \end{array}\right\}$$

This rule allows propagating information about function equivalence by replacing one function with another. This rule is best applied to every call site of the function being replaced at once: in this case we can simply remove the old function from the polyprogram. We call such a procedure *merging by congruence* since the functions are effectively merged.

Deduplication

$$\left\{\begin{array}{l} f(\overline{x_i}) \equiv E \\ f(\overline{x_i}) \equiv E \end{array}\right\} \mapsto \left\{\, f(\overline{x_i}) \equiv E \right\}$$

This rule is used after merging by congruence to remove a coinciding definition of a function. Its only purpose is to reduce memory consumption. The three aforementioned rules together implement *congruence closure* [6] which lies at the heart of equality saturation.

Unfolding of a function consisting of a function call

$$\left\{\begin{array}{l} f(\overline{x_i}) \equiv h(\overline{e_j(\overline{x_i})}) \\ h(\overline{y_j}) \equiv g(\overline{d_k(\overline{y_j})}) \end{array}\right\} \mapsto \left\{\begin{array}{l} f(\overline{x_i}) \equiv h(\overline{e_j(\overline{x_i})}) \\ h(\overline{y_j}) \equiv g(\overline{d_k(\overline{y_j})}) \\ f(\overline{x_i}) \equiv g(\overline{q_k(\overline{x_i})}) \\ \overline{q_k(\overline{x_i}) \equiv d_k(\overline{e_j(\overline{x_i})})} \end{array}\right\}$$

The unfolding rule breaks apart into several simpler rules. We show only one of these rules for brevity. The most interesting thing is that we don't need a separate operation for substitution into an expression ($E\{x \mapsto e\}$) since non-elementary calls play the role of explicit substitutions.

3. Polyprogram bisimulation

Merging by bisimulation is a generalization of the redefinition rule from the Burstall-Darlington framework. Consider the following polyprogram:

$$\begin{array}{l} f() \equiv S(f()) \\ g() \equiv S(h()) \\ h() \equiv S(g()) \end{array}$$

The goal is to infer $f() \equiv g()$. Turns out, this cannot be done directly with the simple rules mentioned above, so we need something more powerful. In this case the definitions

of these functions are not even isomorphic, so we need a more general relation than isomorphism, which we call a bisimulation because it remotely resembles the notion of bisimulation for labeled transition systems.

3.1. The notion of polyprogram bisimulation

First of all, let's give some auxiliary definitions. If θ is a function from $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$ (which we will write just as $\theta : m \to n$) then it can be applied to any functional symbol to permute, omit and duplicate its parameters. We write this application simply as θf and use the following reduction rule:

$$(\theta e_0)(e_1, \ldots, e_n) \rightsquigarrow e_0(e_{\theta(1)}, \ldots, e_{\theta(m)})$$

Definition 5. *A* morphism of functional symbols *from F_1 to F_2 is a function ϕ that maps each functional symbol $f \in F_1$ to a pair $\phi(f) = (\theta, h)$ where $h \in F_2$ and $\theta : arity(h) \to arity(f)$.*

We will often write $\phi(f) = \theta h$ instead of $\phi(f) = (\theta, h)$.

A morphism of functional symbols maps functional symbols, possibly permuting and dropping their parameters. Morphisms of functional symbols can be applied to definitions and polyprograms. If d is a definition then $\phi(d)$ is obtained by replacing all function names f in d with θh, where $\phi(f) = \theta h$, with subsequent normalization with respect to $\rightsquigarrow$. If P is a polyprogram then $\phi(P) = \{\phi(d) \mid d \in P\}$. For example, if $\phi(g) = idg$ and $\phi(f) = \theta h$ where $\theta(1) = 2$ and $\theta(2) = 1$, then $\phi(f(f(x, y), g(x))) = h(g(x), h(y, x))$.

Note that even if P is a polyprogram, $\phi(P)$ is not necessarily a polyprogram, because ϕ may introduce variable duplication in left hand sides of definitions. We call such objects *quasipolyprograms*. However, if ϕ maps every f into a pair (θ, f') such that θ is injective then $\phi(P)$ will be a polyprogram if P is a polyprogram.

By definition, morphism application affects both elementary and non-elementary calls. This actually considerably complicates the theory of polyprogram bisimulations, because the ability of morphisms to rearrange parameters in non-elementary calls requires considering all possible permutations in the bisimulation enumeration algorithm. To simplify things, we use the following semantically equivalent construct instead of a non-elementary call $f(e_1, \ldots, e_n)$:

$$\textbf{case } C(e_1, \ldots, e_n) \textbf{ of } \{\ C(x_1, \ldots, x_n) \to f(x_1, \ldots, x_n)\ \}$$

Since it is not human-readable, $f(e_1, \ldots, e_n)$ will still be used as a syntactic sugar. Note that now morphisms of functional symbols only affect the order of bound variables. Moreover, in subsequent proofs and definitions we will need to consider only three language constructs.

We call two definitions *α-equivalent*, written $d_1 \approx d_2$, if they are equal up to variable renaming. We use the notation $P_1 \lesssim P_2$ if P_1 is a subpolyprogram of P_2 up to α-equivalence of its definitions.

Definition 6. *A* polyprogram bisimulation *over a polyprogram P is a polyprogram B with two morphisms of functional symbols ϕ and ψ such that $\phi(B) \lesssim P$, $\psi(B) \lesssim P$, and if a function $f \in B$ has no definitions then $\phi(f) = \psi(f)$.*

Polyprogram bisimulations are useful for proving equivalence of functions (and subsequently merging them, the transformation we call *merging by bisimulation*) using the following theorem.

Theorem 1. *Let B with ϕ and ψ be a polyprogram bisimulation over P. If for every interpretation ν of B's functions without definitions there is only one model μ that coincides with ν on B's functions without definitions then it is possible to add definitions of the following form into P for each function $f \in B$ without changing the set of P's models:*

$$g(x_{\theta(1)}, \ldots, x_{\theta(m)}) \equiv h(x_{\xi(1)}, \ldots, x_{\xi(n)})$$

where $(\theta, g) = \phi(f)$, $(\xi, h) = \psi(f)$, $m = arity(g)$, $n = arity(h)$.

Proof. Let μ be a model of P. Then there are two models of B: $\mu_\phi(f) = \theta\mu(g)$ where $(\theta, g) = \phi(f)$, and $\mu_\psi(f) = \xi\mu(h)$ where $(\xi, h) = \psi(f)$. But for every function $f \in B$ with no definitions $\phi(f) = \psi(f)$, hence $\mu_\phi(f) = \mu_\psi(f)$. Since for every interpretation ν of B's functions without definitions there is only one model of B, models μ_ϕ and μ_ψ must be equal. This means that for every function $f \in B$, $\theta\mu(g) = \xi\mu(h)$, i.e. $\mu(g)(x_{\theta(1)}, \ldots, x_{\theta(m)}) = \mu(h)(x_{\xi(1)}, \ldots, x_{\xi(n)})$. This is exactly the semantics of the new definitions, so they can be safely added to P, and μ will still be a model of the augmented quasipolyprogram.

Since adding definitions cannot expand the set of models, the assertion of the theorem is proved. □

Not every bisimulation is good enough for merging by bisimulation, because it must also satisfy the model uniqueness property. To test this property some decidable sufficient conditions may be used, like structural and guarded recursion [1], or the presence of ticks [7]. This topic is out of scope of this paper.

3.2. Enumerating bisimulations

In this section we assume that polyprograms are in decomposed form and non-elementary calls are encoded using case expressions. There is an infinite number of polyprogram bisimulations, so we are going to present an algorithm that produces an infinite stream polyprogram bisimulations, but only of some specific form. The algorithm can be informally outlined in the following way: first enumerate *prebisimulations*, quasipolyprograms that are precursors of bisimulations, and then transform each prebisimulation into a polyprogram bisimulation if possible.

Prebisimulations will be quasipolyprograms consisting of *products of definitions* of the original polyprogram. Its idea of a product of two definitions *def-product*(d_1, d_2) is to combine every pair of corresponding functions into a single one with their parameter lists concatenated, e.g.:

$$\begin{aligned}
&\textit{def-product}((f(x) \equiv x), (g(y) \equiv y)) = (\langle f, g\rangle(x, x) \equiv x)\\
&\textit{def-product}((f(x, z) \equiv \mathbf{case}\ h()\ \mathbf{of}\{S(y) \to g(x, y)\}),\\
&\qquad\qquad (f'(x) \equiv \mathbf{case}\ h'(x)\ \mathbf{of}\{S(y) \to g'(y, x)\})) =\\
&\qquad = (\langle f, f'\rangle(x, z, x') \equiv \mathbf{case}\ \langle h, h'\rangle(x')\ \mathbf{of}\{S(y) \to \langle g, g'\rangle(x, y, y, x')\})
\end{aligned}$$

Definition 7. *Let d and d' be two decomposed definitions with the same language construct, the same number of function calls and the same number of variables bound by corresponding patterns. Assume also that corresponding variables in corresponding patterns of the two definitions have coinciding names, and also if the right hand sides have the form x then x is the same for both definitions, but there are no more variable collisions between the definitions (these requirements may be satisfied by applying α-conversion). Then the* product of these definitions *is defined as follows:*

$$\begin{aligned}
&\mathit{def\text{-}product}(f(x_1,\ldots,x_n) \equiv x_i, f'(y_1,\ldots,y_m) \equiv y_j) = \\
&\quad = \langle f, f'\rangle(x_1 \ldots x_n, y_1 \ldots y_{j-1}, x_i, y_{j+1} y_m) \equiv x_i \\
&\mathit{def\text{-}product}(f(\overline{x}) \equiv C(g_1(\overline{x}_1),\ldots,g_n(\overline{x}_n)), \\
&\qquad\qquad f'(\overline{y}) \equiv C(g'_1(\overline{y}_1),\ldots,g'_n(\overline{y}_1))) = \\
&\qquad\qquad\quad = \langle f, f'\rangle(\overline{x},\overline{y}) \equiv C(\langle g_1, g'_1\rangle(\overline{x}_1,\overline{y}_1),\ldots) \\
&\mathit{def\text{-}product}(f(\overline{x}) \equiv \mathbf{case}\ g_0(\overline{x}_0)\ \mathbf{of}\{C_1(\overline{z}_1) \to g_1(\overline{x}_1);\ldots\}, \\
&\qquad\qquad f'(\overline{y}) \equiv \mathbf{case}\ g'_0(\overline{y}_0)\ \mathbf{of}\{C_1(\overline{z'}_1) \to g'_1(\overline{x'}_1);\ldots\}) = \\
&\quad = \langle f, f'\rangle(\overline{x},\overline{y}) \equiv \mathbf{case}\ g_0(\overline{x}_0, \overline{x'}_0)\ \mathbf{of} \\
&\qquad\qquad\qquad \{C_1(\overline{z}_1) \to \langle g_1, g'_1\rangle(\overline{w}_1, \overline{w'}_1\{\overline{z'}_1 \mapsto \overline{z}_1\});\ldots\}
\end{aligned}$$

Here $\langle f, g\rangle$ denotes a unique functional symbol corresponding to f and g with arity $\mathit{arity}(f) + \mathit{arity}(g)$.

We enumerate prebisimulations in the form of trees with back edges growing from a pair of functions. The enumeration procedure is shown in Fig. 1 in the form of a function taking a polyprogram and two functional symbols and returning a set of prebisimulations (programmatically this function should be implemented as returning a stream). Its idea is to traverse the polyprogram P in depth-first order simultaneously from the functions f and f'. A branch may be finished if we encounter a pair of coinciding functions (reflexivity) or a pair of functions which we have already visited (folding). If we do not finish the branch then we choose a pair of definitions of these functions such that the product of these definitions is defined, and descend to pairs of corresponding functions in their right hand sides.

Now let's define two morphisms, $\pi_1(\langle f_1, f_2, l\rangle) = (\gamma_1, f_1)$ where $\gamma_1(i) = i$, and $\pi_2(\langle f_1, f_2, l\rangle) = (\gamma_2, f_2)$ where $\gamma_2(i) = i + \mathit{arity}(f_1)$. A prebisimulation with π_1 and π_2 is not yet a bisimulation for two reasons: it is not a polyprogram because of variable duplication, and π_1 and π_2 differ on functions without definitions.

To transform a prebisimulation into a polyprogram bisimulation duplicated variables should be merged. To do so, we propagate the information about variable equivalence, thus making the quasipolyprogram "coarser". If this process succeeds, the resulting quasipolyprogram may be transformed into a polyprogram, moreover, it will be a bisimulation.

Definition 8. *A quasipolyprogram Q_1 is* no more coarse *than Q_2, written $Q_1 \sqsubseteq Q_2$, if their definitions are in one-to-one correspondence, and for every definition $d_1 \in Q_1$ the corresponding definition d_2 is α-equal to $d_1\{x \mapsto y\}$ for some variables x and y.*

$$
\begin{aligned}
&\mathit{prebisimulations}(P, f, f') = B \\
&\qquad \text{where } (\_, B) = \mathit{prebisimulations}'(P, f, f', \{\}) \\
&\mathit{prebisimulations}'(P, f, f', \mathit{history}) \\
&\quad = \{(f_{new}, \{\}) \mid f = f'\} \\
&\quad \cup \{(f_{old}, \{\}) \mid (f, f', f_{old}) \in \mathit{history}\} \\
&\quad \cup \{(f_{new}, \{q'\} \cup B_1 \cup \ldots \cup B_n) \mid \\
&\qquad\qquad d = (f(\ldots) \equiv L(\ldots)) \in P, \\
&\qquad\qquad d' = (f'(\ldots) \equiv L(\ldots)) \in P, \\
&\qquad\qquad \text{such that } \mathit{def\text{-}product}(d, d') \text{ is defined}, \\
&\qquad\qquad q = \mathit{def\text{-}product}(d, d'), \\
&\qquad\qquad (\langle f, f' \rangle(\ldots) \equiv L(\langle g_1, g_1' \rangle(\ldots), \ldots)) = q, \\
&\qquad\qquad \mathit{history}' = \mathit{history} \cup \{(f, f', f_{new})\}, \\
&\qquad\qquad (h_i, B_i) \in \mathit{prebisimulations}'(P, g_i, g_i', \mathit{history}'), \\
&\qquad\qquad q' \text{ is } q \text{ with } \langle f, f' \rangle \text{ replaced with } f_{new} \\
&\qquad\qquad\qquad \text{and } \langle g_i, g_i' \rangle \text{ replaced with } h_i\} \\
&\quad \text{where } f_{new} = \langle f, f', l \rangle \text{ where } l \text{ is a fresh unique label}, \\
&\quad f_{new} \text{ has arity } \mathit{arity}(f) + \mathit{arity}(f')
\end{aligned}
$$

Figure 1. Enumeration of prebisimulations

Information about variable equivalence may be propagated with the following transformation.

Definition 9. *Let Q be a quasipolyprogram and $f(x_1, \ldots, x_n)$ be a term from some of Q's definitions such that x_i and x_j are one and the same variable. The following transformation is called* variable equivalence propagation step*: for some definition of Q containing a term $f(y_1, \ldots, y_n)$ replace y_j with y_i in the whole definition.*

Example 1. *Consider the following polyprogram:*

$$
\begin{aligned}
g(x, y, z) &\equiv C(f(x, y), g(z, x, y)) \\
f(x, x) &\equiv x
\end{aligned}
$$

Since $f(x, x)$ in the second definition has a variable duplication, it can be propagated to the first definition by replacing y with x, leading to the following definition:

$$g(x, x, z) \equiv C(f(x, x), g(z, x, x))$$

Now $g(x, x, z)$ contains variable duplication, so variable equivalence may be propagated further, resulting in the definition $g(x, x, x) \equiv C(f(x, x), g(x, x, x))$.

Sometimes variable equivalence propagation may lead to variable duplication in patterns (like **case** $h(x)$ **of**$\{C(x, x) \to f(x, x)\}$*). Quasipolyprograms containing such definitions will not lead to bisimulations and may be filtered out.*

If Q' is derived from Q by a variable equivalence propagation step then $Q \sqsubseteq Q'$ by the definition of $\sqsubseteq$. In particular, it means that we can fully propagate variable equivalence in a finite number of steps. Another important property of variable equivalence propagation is that it commutes with $\sqsubseteq$ in the following sense.

Proposition 1. *If $Q_1 \sqsubseteq Q_2$ and it is possible to apply a variable equivalence propagation step to Q_1 and get Q_1' then it is possible to apply no more than one variable equivalence propagation step to Q_2 and get Q_2' such that $Q_1' \sqsubseteq Q_2'$.*

This implies confluence of variable equivalence propagation.

Before performing variable equivalence propagation we need to fix another issue of a prebisimulation: we need to equate corresponding variables of functions without definitions. That is, for each call of function without definitions of the form

$$\langle f, f, l\rangle(x_1, \ldots, x_m, y_1, \ldots, y_m),$$

where $m = \mathit{arity}(f)$, replace y_i with x_i in the whole definition containing this call. This transformation is very similar to a variable equivalence propagation step, and the resulting quasipolyprogram is more or equally coarse than the original prebisimulation.

If variable equivalence is fully propagated then for every term $f(x_1, \ldots, x_n)$ from the quasipolyprogram, if variables x_i and x_j coincide then for every other term $f(y_1, \ldots, y_n)$ variables y_i and y_j also coincide. In this case the following proposition may be applied to transform the quasipolyprogram into a polyprogram.

Proposition 2. *Let Q be a quasipolyprogram. Assume that for each function f of arity n, the set $\{1 \ldots n\}$ of its argument positions can be partitioned into m equivalence classes such that if i and j are from the same class then for every term of the form $f(x_1, \ldots, x_n)$ from Q the variables x_i and x_j coincide. Then there is a pair of morphisms, σ and σ' such that $\sigma'(\sigma(Q)) = Q$. Moreover, if for every term of the form $f(y_1, \ldots, y_n)$ such that variables y_i and y_j coincide, i and j are from the same class, then the quasipolyprogram $\sigma(Q)$ is a polyprogram.*

Proof. For each function $f \in Q$ of arity n with m equivalence classes there is a mapping $\xi_f : n \to m$ that maps each position to the corresponding equivalence class index, and a mapping $\xi_f^{-1} : m \to n$, its right inverse, which maps each equivalence class index to some representative. Let's define $\sigma(f) = (\xi_f^{-1}, f')$ where f' is a function with arity m corresponding to the function f, and $\sigma'(f') = (\xi_f, f)$.

Note that σ' is not a left inverse of σ, so we need to prove that $\sigma'(\sigma(Q)) = Q$. For each definition $d \in Q$, $\sigma'(\sigma(d))$ will replace each occurrence of $f(x_1, \ldots, x_n)$ with $(\xi_f^{-1}\xi_f f)(x_1, \ldots, x_n)$ which reduces to to $f(x_{\xi_f^{-1}\xi_f(1)}, \ldots, x_{\xi_f^{-1}\xi_f(n)})$. Now if $\xi_f^{-1}\xi_f(i) = j$ then i and j are from the same class, so $x_j = x_i$ by the hypothesis of the proposition, and we can rewrite this term as $f(x_1, \ldots, x_n)$ which is equal to the corresponding term in d. Therefore $\sigma'(\sigma(d)) \approx d$, and thus $\sigma'(\sigma(Q)) \approx Q$.

Now assume that for every term from Q of the form $f(y_1, \ldots, y_n)$, if variables y_i and y_j coincide then i and j are from the same class. In this case any such term will be mapped by σ into the term $f'(y_{\xi_f^{-1}(1)}, \ldots, y_{\xi_f^{-1}(m)})$ which cannot contain duplicate variables, because if $y_{\xi_f^{-1}(l)}$ and $y_{\xi_f^{-1}(k)}$ $(l \neq k)$ coincide then $\xi_f(\xi_f^{-1}(l)) = \xi_f(\xi_f^{-1}(k))$, and consequently $l = k$. Then in this case $\sigma(Q)$ is a polyprogram. □

Now let's combine everything into a polyprogram bisimulation enumeration algorithm, expressed as a function returning a set.

Definition 10.

$$\begin{aligned}
&\mathit{bisimulations}(P, f, f') = \\
&\quad = \{\langle B, \pi'_1, \pi'_2\rangle \mid Q \in \mathit{prebisimulations}(P, f, f'),\\
&\qquad\qquad Q' \text{ is } Q \text{ with corresponding variables of functions}\\
&\qquad\qquad\qquad \text{without definitions equated,}\\
&\qquad\qquad Q'' \text{ is } Q' \text{ with variable equivalence fully propagated,}\\
&\qquad\qquad B = \sigma(Q'') \text{ is converted from } Q'' \text{ by Proposition 2,}\\
&\qquad\qquad \text{let } \pi'_1 = \pi_1 \circ \sigma' \text{ and } \pi'_2 = \pi_2 \circ \sigma',\\
&\qquad\qquad B \text{ with } \pi'_1 \text{ and } \pi'_2 \text{ is a polyprogram bisimulation}\}
\end{aligned}$$

Note that we still need to check if the result is a polyprogram bisimulation, because variable equivalence propagation may equate too much, resulting in $\pi'_i(B)$ not being subpolyprograms of P.

The presented algorithm is not strictly complete, since it enumerate bisimulations only of a certain shape, but it is still possible to prove that if there is a bisimulation of this shape, then an equivalent bisimulation will be found by the algorithm.

Theorem 2. *Let P be a polyprogram and B with ϕ and ψ be a polyprogram bisimulation over it such that:*

- *Every functions of B has no more than one definition.*
- *B has a shape of a tree with back edges (as if it was built by depth-first search).*

Let $\phi(s) = (\_, s_\phi)$ and $\psi(s) = (\_, s_\psi)$. Then there is a polyprogram bisimulation $R \in \mathit{bisimulations}(P, s_\phi, s_\psi)$ with ϕ_R and ψ_R such that $\phi(B) \approx \phi_R(R)$ and $\psi(B) \approx \psi_R(R)$.

The proof is omitted for brevity.

Note that the found bisimulation may differ from the original one despite their images being equal. Consider the following polyprogram:

$$\begin{aligned}
f(x) &\equiv S(f(x))\\
g(x) &\equiv S(g(x))
\end{aligned}$$

We can construct a bisimulation $B = \{h(x) \equiv S(h(x))\}$ with morphisms $\phi = \{h \mapsto (id, f)\}$ and $\psi = \{h \mapsto (id, g)\}$ which will lead to the definition $f(x_1) \equiv g(x_1)$ being added to the polyprogram.

But the function *bisimulations* will not return this bisimulation. Indeed, it will find a prebisimulation $D = \{\langle f, g, l\rangle(x, y) \equiv S(\langle f, g, l\rangle(x, y))\}$, but this prebisimulation has variable equivalence information fully propagated, and it will not be changed during conversion to a polyprogram. So the resulting bisimulation will be exactly D with $\pi_1 = \{\langle f, g, l\rangle \mapsto (\{1 \mapsto 1\}, f)\}$ and $\pi_2 = \{\langle f, g, l\rangle \mapsto (\{1 \mapsto 2\}, g)\}$. This bisimulation will lead to the definition $f(x_1) \equiv g(x_2)$ which is better than $f(x_1) \equiv g(x_1)$ because it indicates that the parameters of the both functions are dummy.

3.3. Performance tricks and limitations

The described algorithm is quite inefficient if implemented directly, so in practice it is important to apply some tricks.

- The algorithm was described in a modular way: first enumerate prebisimulations, then try to convert them to bisimulations. In practice it is much more efficient to filter out classes of prebisimulations which cannot be converted to bisimulations before they are fully constructed. To do this we need to add a parameter to the function *prebisimulations* describing the relationship between variables we have inferred so far from the parent pairs of definitions and check if it does not contradict the information we can infer from the pair of definitions we are currently considering to add to the prebisimulation. This will actually perform partial variable equivalence propagation. Note though that we still need to fully propagate variable equivalence in the end since this trick cannot filter out all prebisimulations not leading to bisimulations.

- Since we are only interested in bisimulations for which we can prove model uniqueness, we can also partially check these model uniqueness conditions in the function *prebisimulations*. They should be usually checked during folding to one of the predecessor pairs of functions (e.g. forbid folding if we haven't passed through a constructor or a pattern matching).

- For many pairs of functions it is immediately obvious that we cannot find a bisimulation with a unique model growing from this pair of functions, because they just cannot be equal. This information can be inferred by analyzing certain definitions of these functions, for example if one function has a definition $f(\ldots) \equiv C(\ldots)$, and the other has $g(\ldots) \equiv D(\ldots)$, then they cannot be equal because of different top-level constructors. Another way is to run the functions on some data to find counterexamples to their equivalence.

- Running functions on test data can also help to infer more information about variable equivalence which can be used in conjunction with the first trick.

- The algorithm enumerates an infinite number of bisimulations, which may be useless in practice. Of course, we can limit the depth of our search with $kn^2m!$, where k is the number of definitions, n is the number of functions and m is the maximal arity: all deeper bisimulations will be just equivalent to some more shallow bisimulations. But this is still a big number, so in practice it is better to limit the number of generated prebisimulations and use some depth-limiting heuristics (like limiting the factor of loop unrolling). Note that if all the aforementioned filtering tricks are used, then usually the first found prebisimulation will be a bisimulation with a unique model.

- Sometimes we visit the same pair of functions several times, so we can memoize sets of prebisimulations. It is especially important in the case when we want to find prebisimulations for all pairs of functions. Note though that the set of prebisimulations depends not only on the pair of functions, but also on the history.

4. Related work

Polyprograms are essentially systems of equations from the Burstall-Darlington framework [3]. Decomposed polyprograms are closer to AST and can be used to implement equality saturation [8] for functional languages, which indicates that equality saturation may be seen as another instance of the Burstall-Darlington framework.

Our definition of polyprogram bisimulation is not relational and instead it is based on the notion of a span, although it can be reformulated in relational form. It should be noted that polyprogram bisimulation and LTS bisimulation are quite different since nondeterminism in polyprograms is not related to nondeterminism in LTS. Polyprogram bisimulation also resembles the notion of term graph bisimilarity [2].

Polyprogram bisimulation is used to implement merging by bisimulation, a generalization of the redefinition rule from the Burstall-Darlington framework. Correctness of merging by bisimulation has to be established for each bisimulation by checking additional conditions which are not discussed in this paper. These conditions may be based on termination and productivity conditions [1] as in our previous work, or on the theory of improvement [7] as in the supercompiler of Ilya Klyuchnikov [11].

Our bisimulation enumeration algorithm is related to finding intersection of two languages of term equalities [4]. It is also structurally very similar to supercompilation [9].

5. Conclusion

In this paper we have introduced the notions of a polyprogram and a decomposed polyprogram. A polyprogram is a generalization of a program which admits multiple definitions of a single function. Decomposed polyprograms are polyprograms whose definitions are decomposed into definitions of the simplest form by introducing intermediate functions. Decomposed polyprograms are better suited for reasoning and implementation.

We have presented several main transformation rules in two styles: in the style of the Burstall-Darlington framework for ordinary polyprograms and in the style of equality saturation for decomposed polyprograms. This shows the connection between the two program transformation methods.

We have also introduced the notion of polyprogram bisimulation, on which merging by bisimulation is based. We have presented a bisimulation enumeration algorithm, which enumerates polyprogram bisimulations of a specific form, and proved its correctness.

References

[1] Abel A., Altenkrich T., “A predicative analysis of structural recursion”, *Journal of Functional Programming*, **12**:1 (2002), 1–41.

[2] Ariola Z. M., Klop J. W., Plump D., “Bisimilarity in Term Graph Rewriting”, *Information and Computation*, **156** (2000), 2–24.

[3] Burstall R. M., Darlington J., “A transformation system for developing recursive programs”, *Journal of the ACM*, **24**:1 (1977), 44–67.

[4] Emelyanov P., “Analysis of equality relationships for imperative programs”, *CoRR*, 2006, https://arxiv.org/abs/cs/0609092.

[5] Grechanik S., "Inductive prover based on equality saturation (extended version)", *Proceedings of the Fourth International Valentin Turchin Workshop on Metacomputation*, 2014, 26–53.

[6] Nelson G., Oppen D.C., "Fast decision procedures based on congruence closure", *Journal of the ACM*, **27**:2 (1980), 356–364.

[7] Sands D., "Total correctness by local improvement in the transformation of functional programs", *ACM TOPLAS*, **18**:2 (1996), 175–234.

[8] Tate R., Stepp M., Tatlock Z., Lerner S., "Equality saturation: a new approach to optimization", *SIGPLAN Not.*, **44**:1 (2009), 264–276.

[9] Turchin V., "The concept of a supercompiler", *ACM TOPLAS*, **8**:3 (1986), 292–325.

[10] Scott D.S., "Domains for Denotational Semantics", *Automata, Languages, and Programming: 9th Colloquium Aarhus, Denmark*, Lecture Notes in Computer Science, **140**, 1982, 577–610.

[11] Klyuchnikov I., Romanenko S., "Towards Higher-Level Supercompilation", *Proceedings of the Second International Workshop on Metacomputation in Russia*, 2010, 82–101.

Гречаник С. А., "Полипрограммы и бисимуляция полипрограмм", *Моделирование и анализ информационных систем*, **25**:5 (2018), 534–548.

DOI: 10.18255/1818-1015-2018-5-534-548

Аннотация. Полипрограмма — это обобщение программы, допускающее множественность определений одной и той же функции. Подобные объекты возникают в различных системах преобразования программ, таких как система Бёрстолла–Дарлингтона и насыщение равенствами. В данной работе мы вводим понятие полипрограммы на нестрогом функциональном языке первого порядка. Мы определяем денотационную семантику полипрограмм и описываем некоторые преобразования полипрограмм в двух разных стилях: в стиле системы Бёрстолла–Дарлингтона и в стиле насыщения равенствами. Преобразования в стиле насыщения равенствами осуществляются над полипрограммами в расчленённой форме, в которой стирается грань между функциями и выражениями и между подстановкой и раскрытием вызова функции. Расчленённые полипрограммы хорошо подходят для реализации и проведения рассуждений, но трудны для человеческого восприятия. Мы также вводим понятие бисимуляции полипрограмм, на котором основано преобразование — слияние по бисимуляции, соответствующее доказательству эквивалентности функций по индукции или коиндукции. Бисимуляция полипрограмм — понятие, вдохновлённое понятием бисимуляции размеченных систем переходов, но несколько от него отличающееся, поскольку бисимуляция полипрограмм рассматривает каждое определение как самодостаточное, т.е. функция полипрограммы задаётся любым своим определением, в то время как в размеченной системе переходов поведение системы в состоянии определяется всей совокупностью переходов, которые можно осуществить из этого состояния. Мы предлагаем алгоритм перечисления бисимуляций некоторого определённого вида. Алгоритм состоит из двух фаз: перечисление пребисимуляций и преобразование их в бисимуляции. Такое разделение требуется из-за того, что бисимуляции полипрограмм учитывают возможность перестановки параметров функций. Мы доказываем корректность данного алгоритма, а также формулируем некоторую слабую форму его полноты. Статья публикуется в авторской редакции.

Ключевые слова: полипрограммы, преобразование программ, насыщение равенствами, бисимуляция

Об авторах:
Гречаник Сергей Александрович, orcid.org/0000-0001-8575-9689, канд. физ.-мат. наук,
Институт прикладной математики им. М.В. Келдыша РАН,
Миусская пл., 4, г. Москва, 125047 Россия, e-mail: sergei.grechanik@gmail.com

Благодарности:
Работа выполнена при поддержке гранта РФФИ №18-31-00412

Моделирование и анализ информационных систем. Т. 25, № 5 (2018), с. 549–560
Modeling and Analysis of Information Systems. Vol. 25, No 5 (2018), pp. 549–560

DOI: 10.18255/1818-1015-2018-5-549-560

UDC 519.68

Etude on Recursion Elimination

Shilov N. V.

Received September 26, 2018

Abstract. Transformation-based program verification was a very important topic in early years of theory of programming. Great computer scientists contributed to these studies: John McCarthy, Amir Pnueli, Donald Knuth ... Many fascinating examples were examined and resulted in recursion elimination techniques known as *tail-recursion* and *co-recursion*. In the paper, we examine just a single example (but new we hope) of recursion elimination via program manipulations and problem analysis. The recursion pattern of the example matches descending dynamic programming but is neither tail-recursion nor co-recursion pattern. Also, the example may be considered from different perspectives: as a transformation of a descending dynamic programming to ascending one (with a fixed-size static memory), or as a proof of the functional equivalence between recursive and iterative programs (that can later serve as a case-study for automatic theorem proving), or just as a fascinating algorithmic puzzle for fun and exercising in algorithm design, analysis, and verification. The article is published in the author's wording.

Keywords: recursive and standard program schemata, recursive and iterative programs, functional equivalence of programs and program schemata, ascending and descending dynamic programming, recursion elimination, static and dynamic memory, associative and standard arrays

For citation: Shilov N. V., "Etude on Recursion Elimination", *Modeling and Analysis of Information Systems*, **25**:5 (2018), 549–560.

On the authors:
Nikolay V. Shilov, orcid.org/0000-0001-7515-9647, PhD,
Autonomous noncommercial organization of higher education "Innopolis University"
1 Universitetskaya str., Innopolis, Tatarstan Republic, 420500, Russia, e-mail: shiloviis@mail.ru

1. Introduction

1.1. McCarthy 91 function

We would like to start with a short story about the *McCarthy 91 function* that follows (in principle) the corresponding article [21] "*From Wikipedia, the free encyclopedia*".

The function $M : \mathbb{N} \to \mathbb{N}$ is a recursive function, defined by John McCarthy[1] as a test case for formal verification within computer science. The function is defined as

$$M(n) = \begin{cases} n - 10, \text{ if } n > 100; \\ M(M(n + 11)), \text{ if } n \leq 100. \end{cases}$$

[1]Maybe *the only* Turing Laureate that was employed in Soviet Academy of Sciences, namely *Novosibirsk Computing Center* (http://ershov-arc.iis.nsk.su/archive/eaindex.asp?lang=1&did=20184&_ga=1.44945571.687493938.1476117474, accessed September 26, 2018).

The results of evaluating the function are given by

$$M(n) = \begin{cases} n - 10, \text{ if } n > 101; \\ 91, \text{ if } n \leq 101. \end{cases} \tag{1}$$

The function was introduced in papers published by Zohar Manna, Amir Pnueli and John McCarthy in 1970 [16, 15]. These papers represented early developments towards the application of formal methods to program verification. The function has a "complex" recursion pattern (contrasted with simple patterns, such as *recurrence*, *tail-recursion* or *co-recursion*).

Nevertheless the McCarthy 91 function can be computed by an iterative algorithm (program). Really, let us consider an auxiliary recursive function $M_{aux} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$M_{aux}(n, m) = \begin{cases} n, \text{ if } m = 0; \\ M_{aux}(n - 10,\ m - 1), \text{ if } n > 100 \text{ and } m > 0; \\ M_{aux}(n + 11,\ m + 1), \text{ if } n < 100 \text{ and } m > 0. \end{cases}$$

Then $M(n) = M_{aux}(n, 1)$ because of $M_{aux}(n, m) = M^m(n) = \underbrace{M(\ldots M}_{m-times}(n)\ldots)$ for all $m, n \in \mathbb{N}$ (assuming that $M^0 = (\lambda n \in \mathbb{N}.n)$). Since definition of M_{aux} matches tail-recursion pattern then the McCarthy 91 function can be computed by an iterative algorithm/program (and even by a very efficient *iteration-free* algorithm (1)). A formal derivation of an iterative version from the recursive one was given in [20] in 1980 based on the use of continuations.

As the field of Formal Methods advanced, this example appeared repetitively in the research literature. In particular, it is viewed as a "challenge problem" for automated program verification. Donald Knuth generalized the function to include additional parameters [11], formal proofs (using ACL2 theorem prover) that Knuth's generalized function is total can be found in [4, 5].

1.2. Hull Strength Puzzle

We started with a short story about the McCarthy function because we would like to justify our interest to study of translation of other examples functional/recursive programs into iterative algorithms/programs in general and the following problem[2] that we call in the sequel *Hull Strength Puzzle* (HSP).

> *Let us characterize the mechanical stability (strength) of a hull of a mobile phone by an integer h that is equal to the height (in meters) safe for the case to fall down, while height $(h + 1)$ meters is unsafe (i.e. the brick breaks). You have to determine the stability of hulls of a particular kind by dropping them from different levels of a tower of H meters. (One may assume that mechanical stability does not change after a safe fall.) How many times do you need to drop hulls, if you have 2 hulls in the stock? What is the optimal number (of droppings) in this case?*

[2]The problem formulation is just a literary version of the formulation of the *Dropping Bricks Problem* used in [18, 19], another variant of the problem formulation — *Egg dropping puzzle* — can be found in Wikipedia article on *Dynamic Programming* at https://en.wikipedia.org/wiki/Dynamic_programming#Egg_dropping_puzzle (accessed September 26, 2018).

Basically, the question to answer is how to compute the optimal number of droppings G_H, if the height of the tower is H and you have 2 bricks in the stock.

Our purpose is to prove that the problem is solved by the following simple formula

$$G(H) = \arg\min n : \ \frac{n \times (n+1)}{2} \geq H \tag{2}$$

that can be implemented as a trivial non-recursive function (i.e. with iterative body) $G_{iter}(H : \mathbb{N})$:

1. $var\ n : \mathbb{N}$;
2. $n := 0$;
3. $while\ \frac{n\times(n+1)}{2} < H\ do\ n := \ n + 1$;
4. $G_{iter} := n$.

With a purpose to get the above formula (2), let us start with a recursive solution for HSP. This problem is an example of optimization problems. Any optimal method to define the mechanical stability should start with some step (command) that prescribes to drop the first phone from some particular (but optimal) level h. Hence the following equality holds for this particular level h:

$$G_H = 1 + \max\{(h-1), G_{H-h}\},$$

where (in the right-hand side)

1. 1+ corresponds to the first dropping,
2. $(h-1)$ corresponds to the case when the hull of the first phone breaks after the first dropping (and we have to drop the remaining second phone from the levels 1, 2, $\ldots (h-1)$ in a series),
3. G_{H-h} corresponds to the case when the hull of the first phone is safe after the first dropping (and we have to define stability by dropping the pair of phones from $(H-h)$ levels in $[(h+1)\ldots H]$),
4. 'max' corresponds to the worst in two cases above.

Since the particular value h is *optimal*, and *optimality* means *minimality*, the above equality transforms to the following one:

$$G_H = \min_{1\leq h\leq H}(1 + \max\{(h-1), G_{H-h}\}) = 1 + \min_{1\leq h\leq H} \max\{(h-1), G_{H-h}\}.$$

Besides, we can add one obvious equality $G_0 = 0$.

Remark that the sequence of integers G_0, G_1, ... G_H, ... that meet these two equalities is unique since G_0 is defined explicitly, G_1 is defined by G_0, G_2 is defined by G_0 and G_1, G_H is defined by G_0, G_1, ... G_{H-1}. Hence it is possible to move from the sequence G_0,

G_1, ... G_H, ..., to a function $G : \mathbb{N} \to \mathbb{N}$ that maps every natural x to G_x and satisfies the following *functional equation* for the *objective function* G:

$$G(x) \;=\; if\ x = 0\ then\ 0\ else\ 1 + \min_{1 \le h \le x} \max\{(h-1), G(x-h)\}. \tag{3}$$

This equation has a unique solution as it follows from the uniqueness of the sequence G_0, G_1, ... G_H, ... Let us summarize the above discussion as the following proposition.

Proposition 1. *Functional equation (3) has unique solution in* $\mathbb{N}^{\mathbb{N}}$.

Moreover we can go further: the equation (3) can be adopted as a recursive definition of a function, i.e. a recursive algorithm presented in a functional pseudo-code.

1.3. A Special Case of Dynamic Programming

Dynamic Programming was introduced by Richard Bellman in the 1950s [2] to tackle optimal planning problems. At this time, the noun *programming* had nothing in common with more recent *computer programming* and meant *planning* (compare: *linear programming*). The adjective *dynamic* points out that Dynamic Programming is related to a *change of state* (compare: *dynamic logic*, *dynamic system*). *Bellman equation* is a recursive functional equality for the objective function that expresses the optimal solution at the "current" state in terms of optimal solutions at next (changed) states. It formalizes a so-called *Bellman Principle of Optimality*: an optimal program (or plan) remains optimal at every stage.

After analysis of Bellman equations for particular problems [6] several versions of a *(recursive template for/of) (descending) dynamic programming* were suggested and examined. In the present paper we use the most recent and general one [19]:

$$G(x) \;=\; if\ p(x)\ then\ f(x)\ else\ g\Big(x,\ \Big\{h_i\big(x, G(t_i(x))\big),\ i \in [1..n(x)]\Big\}\Big). \tag{4}$$

We consider the template as a *recursive program scheme* [9, 12, 17], i.e. a recursive control flow structure with *uninterpreted symbols*:

- G is the *main* functional symbol representing (after interpretation of *base* functional and predicate symbol) the objective function $G : X \to Y$ for some X and Y;
- p is a basic predicate symbol representing (after interpretation) some *known*[3] predicate $p \subseteq X$;
- f is a basic functional symbol representing (after interpretation) some known[3] function $f : X \to Y$;
- g is a basic functional symbol representing (after interpretation) some known[3] function $g : X \times Z^* \to X$ for some appropriate Z (with a variable arity $n(x) : X \to \mathbb{N}$);

[3] i.e. that we know how to compute

- all h_i and t_i $(i \in [1..n(x)])$ are basic functional symbols representing (after interpretation) some known[3] function $h_i : X \times Y \to Z$, $t_i : X \to X$ $(i \in [1..n(x)])$.

In the sequel do not make an explicit distinction in notation for symbols and interpreted symbols but just verbal distinction by saying, for example, *symbol* g and *function* g.

Equation (3) for Hull Strength Puzzle is a particular example of functional equation that matches the recursive template for descending dynamic programming (4). In the case we have:

- predicate $\lambda x.(x = 0)$ is interpretation for p,
- constant function $\lambda x.0$ is interpretation for f,
- identical function $\lambda x.x$ is interpretation for the arity n,
- for every $i \in [1..n(x)]$, function $\lambda x.(x - i)$ is interpretation for t_i,
- for every $i \in [1..n(x)]$, function $\lambda t.\max\{(i-1), t\}$ is interpretation for h_i,
- function $\lambda x.\lambda w_1 \ldots \lambda w_n.(\min_{1 \le i \le x} w_i)$ is interpretation for g.

A natural question arises: maybe there exists a *standard scheme* [9, 12, 17] (i.e. a flowchart with uninterpreted predicate and functional symbols instead of predicate and functions) that is *functionally equivalent* to recursive scheme (4)? Unfortunately, in general case the answer is negative according to the following proposition proved by M.S. Paterson and C.T. Hewitt [12, 17].

Proposition 2. *The following special case of the recursive template of descending dynamic programming*

$$F(x) = \ if\ p(x)\ then\ x\ else\ f(F(g(x)), F(h(x)))$$

is not equivalent to any standard program scheme (with fix-size static memory).

This proposition does not mean that (potentially) unbounded memory (e.g. system stack or dynamic heap) is *always* required; it just says that for *some* interpretations of *uninterpreted* symbols p, f, g and h the size of required memory depends on the input data. But if p, f, g and h are *interpreted*, it may happen that function F can be computed by an iterative program without unbounded memory. For example, Fibonacci numbers

$$Fib(n) = \ if\ (n = 0\ or\ n = 1)\ then\ 1\ else\ Fib(n-2) + Fib(n-1)$$

matches the pattern of scheme in the above proposition 2, but just three integer variables suffice to compute it by an iterative program.

Thus proposition 2 rules out an opportunity to get iterative solution for Hull Strength Puzzle by specialization [8, 10] of a standard program scheme equivalent to the recursive scheme (4). But this proposition does not prohibit existence of an iterative algorithm for HSP that uses interpreted functions and predicates.

2. Iterative Algorithm for HSP

2.1. Toward Iterative Algorithm

Let us present some (not very formal) derivation of the formula (2) for Hull Strength Puzzle and start with a look at Fig. 1 that depicts an initial part of the graph of G computed according to (3). One can observe that

(monotonicity): G is a non-decreasing function,

(jump property): it has no jumps greater 1.

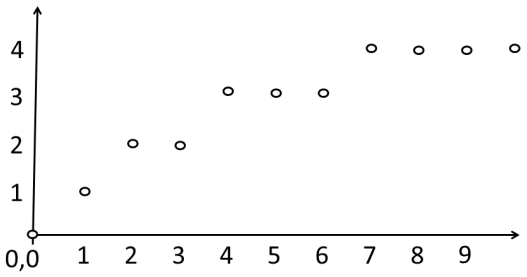


Fig. 1. First values of the function G

Basically these *monotonicity property* and *jump property* follow from semantics of the function G as a solution for HSP, but *we do not know how to prove them formally from the equation (3).*

Then let us proceed symbolically $G(x)$ according to recursive algorithm (3):

$$\begin{aligned} G(x) \;=\; 1 \;+\; \min_{1\le h\le x} \max\{(h-1),\; G(x-h)\} \;= \\ =\; 1 \;+\; \min\Big\{ & \max\{0,\; G(x-1)\},\; \max\{1,\; G(x-2)\}, \\ & \ldots\ldots\ldots\ldots\ldots\ldots \\ & \max\{(y-2),\; G(x-y-1)\}, \\ & \max\{(\mathbf{y-1}),\; \mathbf{G(x-y)}\}, \\ & \max\{(y,\; G(x-y+1)\}, \\ & \ldots\ldots\ldots\ldots\ldots\ldots \\ & \max\{(x-2),\; G(1)\},\; \max\{(x-1),\; G(0)\}\Big\} \end{aligned}$$

where line in **bold** $\max\{(\mathbf{y-1}),\; \mathbf{G(x-y)}\}$ corresponds to the last value $h \in [1..x]$ such that $(h-1) \le G(x-h)$. Due to the monotonicity property we have

$$G(x) \;=\; 1 + G(x-y). \tag{5}$$

Due to monotonicity and jump properties we have

$$\left[\begin{array}{l} \text{either } G(x-y) = y \\ \text{or } G(x-y) = (y-1) \end{array}\right. ;$$

let us accept the late option and rule out the former (*but we can not prove why we may do it*):

$$G(x-y) \;=\; (y-1). \tag{6}$$

Now, for the technical convenience, let a be $(x-y)$, b — $(y-1)$; then $x=(a+b+1)$ and (5) and (6) lead to the equality

$$G(a)=b \text{ inplies } G(a+b+1) \;=\; (b+1). \tag{7}$$

Together with another equality $G(0)=0$ it leads to the following equality (that can be proved by induction)

$$G(\sum_{h=1}^{h=n} h) \;=\; n. \tag{8}$$

2.2. An Optimal Procedure for Mechanical Strength

Formula (8) leads also to the following procedure $Strength(Hight : \mathbb{N})$ to define mechanical strength of the hull using 2 identical mobile phones (named the first and the second in the sequel):

1. $var\ n,\ step,\ Current,\ Next : \mathbb{N}$;
2. let $n := \arg\min n : \frac{n\times(n+1)}{2} \geq H$;
3. let $step := n$ and $Current := 0$;
4. dropping the first phone:
   while $Current < Hight$ and $step > 0$ do
   (a) let $Next := \min\{Hight,\ (Current + step)\}$
   and drop the first phone from the $Next$ level;
   (b) if the drop was unsafe (i.e. the hull of the first phone breaks)
   then break the loop and go to 5 (dropping the second phone);
   (c) let $Current := Next$ and $step := (step-1)$;
5. dropping the second phone:
   let $Current := (Current+1)$;
6. while $Current < Next$ do
   (a) drop the second phone from the $Current$ level;
   (b) if the drop was unsafe (i.e. the hull of the second phone breaks)
   then break the loop and go to 7 (report the strength);
   (c) let $Current := (Current+1)$;
7. report the strength: $(Current-1)$.

To explain the idea of the procedure $Strength(H)$ (where $H \in \mathbb{N}$), let us assume that the height of the tower H is exactly the sum of an arithmetic progression n, $(n-1)$, ... (2), 1. Then the procedure divides the tower onto n layers of heights $step_1 = n$, $step_2 = (n-1)$, ... $step_{(n-1)} = 2$ and $step_n = 1$. (For example, in the left part of Fig. 2 one can see a tower of hight 10 divided on 4 layers of heights 4, 3, 2 and 1.)

The first loop in the procedure prescribes to drop the first phone in a sequence (while it is safe) from the (top of) layers at levels n, $\big(n+(n-1)\big)$, $\Big(\big(n+(n-1)\big)+(n-2)\Big)$, ... until it breaks after dropping from the top of some layer $k \geq 1$ from the level $\Big(\ldots\Big(\big(n+(n-1)\big)+(n-2)\Big)\cdots+(n-(k-1))\Big)$. (In the exercise of the procedure in the right part of Fig. 2 $k = 2$.)

The second loop in the procedure prescribes to use the second phone moving one by one (while the phone is safe) all levels from $\Big(\ldots\Big(\big(n+(n-1)\big)+(n-2)\Big)\cdots+1\Big)$ to $\Big(\ldots\Big(\big(n+(n-1)\big)+(n-2)\Big)\cdots+(n-(k-2))\Big)$ of the layer $k \geq 1$ (from the top of which the first phone fell down and broke). (In the exercise of the procedure in the right part of Fig. 2 two levels — 5 and 6 — were examined.)

The mechanical strength of the hull is the last level from which the second brick was safely dropped. (In the exercise of the procedure in the right part of Fig. 2 it is level 5.)

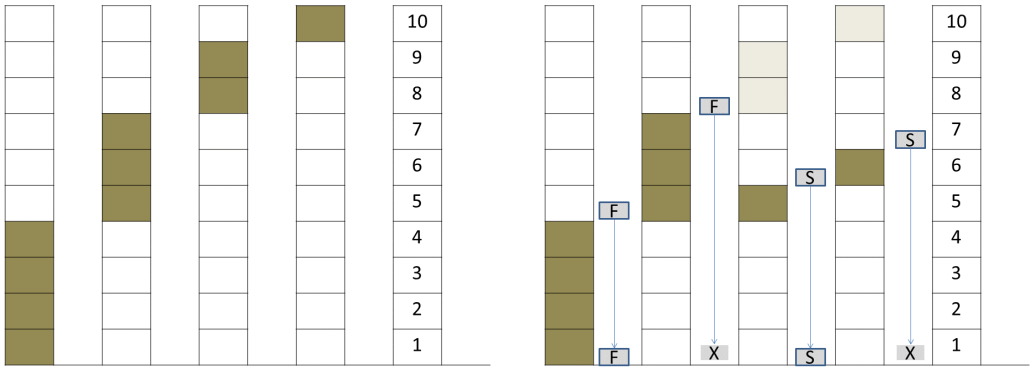


Fig. 2. The layers (left) and an exercise (right) of the procedure *Strength* for the tower of height 10 and two bricks (F and S) of mechanical strength 5

Optimality (i.e. the minimality of droppings) of the procedure *Strength* can be proved by contradiction. Really, let us assume in contrary that for some $H \in \mathbb{N}$ another method gives better number m of droppings in the worst case. Better number this time means that $m < n = \arg\min n : \frac{n\times(n+1)}{2} \geq H$. This method also divide the tower on layers from top of which the first phone have to be dropped (according to the method): assuming $Level_0 = 0$,

- the top of the first layer is $Level_1 = Level_0 + m_1$,
- the top of the second layer is $Level_2 = (Level_1 + m_2)$,

- .....
- the top of the last layer is $Level_{last} = Level_{last-1} + m_{last} = H$.

Remark that $last \leq m$, because the first phone can survive all m droppings. Then we have:

- $m_1 \leq m$, because the first phone can break after the *first* dropping;
- $m_2 \leq (m-1)$, because the first phone can break after the *second* dropping;
- .....
- $m_k \leq (m-(k-1))$, because the first phone can break after k-*th* dropping, $(1 \leq k \leq last)$;
- .....
- $m_{last} \leq (m-(last-1))$, because the first phone can break after the *last* its dropping.

Hence we have:

$$H = \\ = m_1 + m_2 + \ldots m_{last} \leq m + (m-1) + \ldots (m-(last-1)) \leq \sum_{k=1}^{k=m} k \leq \\ \leq \sum_{k=1}^{k=n} k;$$

at the same time (according to choice of n)

$$n = \arg\min n : \ \frac{n \times (n+1)}{2} \geq H;$$

it implies that $m = n$. — Contradiction with the assumption $m < n$.

Thus we prove the following proposition.

Proposition 3. *Procedure Stregth implements an optimal (in sense of number of droppings) method to define mechanical strength of bricks using 2 bricks: for any given $H \in \mathbb{N}$ it defines mechanical strength dropping bricks*

$$\arg\min n : \ \frac{n \times (n+1)}{2} \geq H$$

times at most (and this upper bound is exact).

According to proposition 1, functional equation (3) has unique solution in $\mathbb{N}^{\mathbb{N}}$ that computes the optimal number of droppings that is sufficient to define the strength. Due to this uniqueness and according to proposition 3, this solution is defined by equality (2.) and $G = G_{iter}$.

3. Conclusion: Towards Formal Verification

Let us start with a summary of a contribution of this paper.

- The paper discusses a so-called Hull Strength Puzzle (see subsection 1.2.) and how to eliminate recursion and build an iterative algorithm to solve the problem.
- The problem under study is an instance of so-called *learning problem* to determine the function in some family that has certain properties by testing (querying) the function several times.
- The recursive solution of the problem is a particular instance of dynamic programming and matches descending dynamic programming template (see subsection 1.3.).
- Unfortunately, the descending dynamic programming template is not equivalent to any fixed standard program scheme (see subsection 1.3.) and hence iterative solution for the problem can not result from a general one by program specialization [8, 10].
- Also, the recursive solution matches neither tail-recursion nor recurrent pattern that can be converted into iterative algorithms by well-known techniques [11].
- We derived a candidate for iterative solution for Hull Strength Puzzle by some program manipulations (basically, loop unfolding) and (not-very sound) semantic analysis of the unfolded loop (see subsection 2.1.).
- Finally we give (see subsection 2.2.a round-about (and very much) human-oriented proof of correctness of the iterative algorithm for Hull Strength Puzzle (using an optimal method to define mechanical strength of the bricks).

Some topics for further studies are presented below (from the nearest to that which require more time).

- To prove using a proof-assistance (ACL2 most probably) that iterative and recursive definitions for the function G (see subsection 1.2.) are equivalent.
- To investigate how to generalize the pattern of the recursive function and very particular manipulations used/presented in this paper for recursion elimination in more general cases.
- Investigate methods to find recursive patterns admitting recursion elimination. Maybe, machine learning can help to advance in this direction.
- To design and implement a plugin for some IDE (Integrated Development Environment) that analyses program code to find recursive patterns admitting recursion elimination and eliminates these cases of recursion at object code level.

We would like to conclude with some references to related research. Transformation approach to efficient programs is under study for high-level functional programming languages [7]. Use of integer arrays for efficient recursion elimination for functions of integer argument was suggested first (up to our knowledge) in [3] and use of auxiliary (associative) arrays for more general recursion elimination was studied later in [13], and more broadly in [14].

References

[1] Olver P. J., *Applications of Lie groups to differential equations*, Springer-Verlag, New York, 1993.

[2] Bellman R., "The theory of dynamic programming", *Bulletin of the American Mathematical Society*, **60** (1954), 503–516.

[3] Berry G., "Bottom-up computation of recursive programs", *RAIRO — Informatique Théorique et Applications (Theoretical Informatics and Applications)*, **10**:3 (1976), 47–82.

[4] Cowles J., *Computer-Aided reasoning: ACL2 case studies*, Kluwer Academic Publishers, 2000.

[5] Cowles J., Gamboa R., "Contributions to the Theory of Tail Recursive Functions", 2004, http://www.cs.uwyo.edu/~ruben/static/pdf/tailrec.pdf, accessed September 26, 2018.

[6] Cormen T. H. et al., *Introduction to Algorithms (3rd ed.)*, The MIT Press, 2009.

[7] De Moor O., Sittampalam G., "Generic Program Transformation", Lecture Notes in Computer Science, **1608**, 1999, 116–149.

[8] Ershov A. P., "Mixed computation: potential applications and problems for study", *Theor. Comp. Sci.*, **18**:1 (1982), 41–67.

[9] Greibach S. A., *Theory of Program Structures: Schemes, Semantics, Verification*, Lecture Notes in Computer Science, **36**, Springer, 1975.

[10] Jones N. D. et al., *Partial Evaluation and Automatic Program Generation*, Prentice Hall International, 1993.

[11] Knuth D. E., *Textbook Examples of Recursion*, 1991, arXiv:cs/9301113[cs.CC], accessed September 26, 2018.

[12] Kotov V. E., Sabelfeld V. K., *Theoriya Schem Programm*, Nauka, 1991.

[13] Liu Y. A., Stoller S. D., "Program optimization using indexed and recursive data structures", *Proceedings of the 2002 ACM SIGPLAN workshop on Partial evaluation and semantics-based program manipulation*, 2002, 108–118.

[14] Liu Y. A., *Systematic Program Design: From Clarity to Efficiency*, Cambridge University Press, 2013.

[15] Manna Z., Pnueli A., "Formalization of Properties of Functional Programs", *Journal of the ACM*, **17**:3 (1970), 555–569.

[16] Manna Z., McCarthy J., "Properties of programs and partial function logic", *Machine Intelligence*, **5** (1970), 79–98.

[17] Paterson M. S., Hewitt C. T., "Comperative Schematology", *Proc. of the ACM Conf. on Concurrent Systems and Parallel Computation*, 1970, 119–127.

[18] Shilov N. V., "Unifying Dynamic Programming Design Patterns", *Bulletin of the Novosibirsk Computing Center (Series: Computer Science)*, **34** (2012), 135–156.

[19] Shilov N. V., "Algorithm Design Patterns: Program Theory Perspective", *Proc. of Fifth Int. Valentin Turchin Workshop on Metacomputation (META-2016)*, 2016, 170–181.

[20] Wand M., "Continuation-Based Program Transformation Strategies", *Journal of the ACM*, **27**:1 (1980), 164–180.

[21] “McCarthy 91 function”, https://en.wikipedia.org/wiki/McCarthy_91_function, accessed September 26, 2018.

Шилов Н. В., "Этюд об устранении рекурсии", *Моделирование и анализ информационных систем*, **25**:5 (2018), 549–560.

DOI: 10.18255/1818-1015-2018-5-549-560

Аннотация. Трансформационный подход к верификации программ был очень популярной темой исследований в первые десятилетия теории программирования. Многие выдающиеся пионеры теории программирования внесли свой вклад в разработку данного направления исследований: Джон Маккарти, Амир Пнуели, Дональд Кнут ... Много интересных примеров трансформационного подхода было тщательно изучено, что привело к методам устранения рекурсии, известным как *хвостовая рекурсия* и как *ко-рекурсия*. В данной работе мы подробно исследуем (мы надеемся, новый) пример устранения рекурсии, основанный на трансформациях программы и анализе задачи, решаемой этой программой. Наш пример является частным случаем нисходящего динамического программирования, но не является ни примером хвостовой рекурсии, ни ко-рекурсии. Этот пример можно рассмотреть с разных точек зрения: как пример преобразования нисходящего динамического программирования к восходящему (с использованием только статической памяти фиксированного размера), или как доказательство функциональной эквивалентности между рекурсивной и итеративной программами (которое в дальнейшем может послужить примером для автоматического доказательства), или как захватывающую алгоритмическую головоломку либо задачу дизайна, анализа и верификации алгоритмов. Статья публикуется в авторской редакции.

Ключевые слова: рекурсивные и стандартные схемы программ, рекурсивные и итеративные программы, функциональная эквивалентность программ и схем программ, восходящее и нисходящее динамическое программирование, устранение рекурсии, ассоциативные и стандартные массивы, статическая и динамическая память

Об авторах:
Шилов Николай Вячеславович, orcid.org/0000-0001-7515-9647, канд. физ.-мат. наук, доцент,
Автономная некоммерческая организация высшего образования “Университет Иннополис”,
ул. Университетская, 1, г. Иннополис, Республика Татарстан, 420500, Россия, e-mail: shiloviis@mail.ru

Моделирование и анализ информационных систем. Т. 25, № 5 (2018), с. 561–571
Modeling and Analysis of Information Systems. Vol. 25, No 5 (2018), pp. 561–571

Теория автоматов
Automata Theory

DOI: 10.18255/1818-1015-2018-5-561-571

UDC 519.713

Universal Hypergraphic Automata Representation by Autonomous Input Symbols

Khvorostukhina E. V., Molchanov V. A.

Received July 25, 2018

Abstract. Hypergraphic automata are automata with state sets and input symbol sets being hypergraphs which are invariant under actions of transition and output functions. Universally attracting objects of a category of hypergraphic automata are automata $\mathrm{Atm}(H_1, H_2)$. Here, H_1 is a state hypergraph, H_2 is classified as an output symbol hypergraph, and $S = \mathrm{End}\, H_1 \times \mathrm{Hom}(H_1, H_2)$ is an input symbol semigroup. Such automata are called universal hypergraphic automata. The input symbol semigroup S of such an automaton $\mathrm{Atm}(H_1, H_2)$ is an algebra of mappings for such an automaton. Semigroup properties are interconnected with properties of the algebraic structure of the automaton. Thus, we can study universal hypergraphic automata with the help of their input symbol semigroups. In this paper, we investigated a representation problem of universal hypergraphic automata in their input symbol semigroup. The main result of the current study describes a universal hypergraphic automaton as a multiple-set algebraic structure canonically constructed from autonomous input automaton symbols. Such a structure is one of the major tools for proving relatively elementary definability of considered universal hypergraphic automata in a class of semigroups in order to analyze interrelation of elementary characteristics of universal hypergraphic automata and their input symbol semigroups. The main result of the paper is the solution of this problem for universal hypergraphic automata for effective hypergraphs with p-definable edges. It is an important class of automata because such an algebraic structure variety includes automata with state sets and output symbol sets represented by projective or affine planes, along with automata with state sets and output symbol sets divided into equivalence classes. The article is published in the authors' wording.

Keywords: automaton, semigroup, hypergraph, input symbol

For citation: Khvorostukhina E. V., Molchanov V. A., "Universal Hypergraphic Automata Representation by Autonomous Input Symbols", *Modeling and Analysis of Information Systems*, **25**:5 (2018), 561–571.

On the authors:
Ekaterina V. Khvorostukhina, orcid.org/0000-0002-2775-5732, PhD,
Yuri Gagarin State Technical University of Saratov,
77 Politechnicheskaya str., Saratov 410054, Russia, e-mail: khvorostukhina85@gmail.com

Vladimir A. Molchanov, orcid.org/0000-0001-6509-3090, PhD,
Saratov State University,
83 Astrakhanskaya str., Saratov, 410012, Russia, e-mail: molchanovva@mail.ru

Introduction

Automata theory is among major computer science branches studying data conversion devices. Such devices arise in control theory tasks, communication theory problems,

economic logistics tasks, and others. A mathematical model of data conversion device is automaton $A = (X, S, Y, \delta, \lambda)$. It is a multiple-set algebra consisting of three sets X, S, Y and two binary operations $\delta : X \times S \to X$, $\lambda : X \times S \to Y$. Here X is called a state set, S is classified as an input symbol set, Y is an output symbol set, δ is a transition function, and λ is an output function. The transition function δ for each input symbol $s \in S$ defines the state $\delta(x, s)$, in which the automaton moves from the state $x \in X$ depending on the symbol s. Similarly, the output function λ for each input symbol $s \in S$ determines the output symbol $\lambda(x, s)$. The symbol $\lambda(x, s)$ is generated by the automaton in the state $x \in X$ depending on the symbol s. Thus, for each fixed input symbol $s \in S$ the automaton A determines the transition function $\delta_s : X \to X$ and an output function $\lambda_s : X \to Y$ by the formulas: $\delta_s(x) = \delta(x, s)$ and $\lambda_s(x) = \lambda(x, s)$. For the elements $s, t \in S$, sequential action of transition functions δ_s, δ_t defines associative composition of input symbols $s \cdot t$ so that $\delta_{s \cdot t} = \delta_s \delta_t$.

Therefore, it is often presumed that the input symbol set S is a semigroup interrelated with the transition function and output function of the automaton A by the following formulas: $\delta(x, s \cdot t) = \delta(\delta(x, s), t)$, $\lambda(x, s \cdot t) = \lambda(\delta(x, s), t)$ for any $x \in X$, $s, t \in S$. We also denote the semigroup S as $\mathrm{Inp}(A)$.

Depending on the specifics of the computer science tasks, we can model data conversion device as structured automaton. Its state set X and output symbols set Y are algebraic structures which are invariant under actions of transition and output functions of such automaton. Examples of such structures include a probability space structure, a linear space structure, a topological space structure, an ordered set structure, etc. (see e.g. [1]). Thus, well-known specific computer science tasks lead to the notions of a probability automaton, a linear automaton, a topological automaton, and an ordered automaton. Many authors studied such automata (e.g., [1], [2], [3], [4]). In this approach structured automaton is a focus of scientific interest and current studies of algebraic automata theory, which is an important universal algebra branch. Also, it has a variety of applications to combinatorial automata investigations connected with automaton behavior, analysis and synthesis of automata, as well as to formal language theory, algorithm theory, and many other computer science branches [1], [5].

In the paper we continued to study this field. We investigated algebraic properties of hypergraph automata, i.e. automata with state sets and input symbol sets being hypergraphs [6]. Automata under our study form a wide and important class of automata because a hypergraph is a generalization of such concepts as graph, set partition, plane [7] and others. Thus, such algebraic structure variety includes automata with state sets and output symbol sets represented by planes, along with automata with state sets and output symbol sets divided into equivalence classes.

The main focus of our research is universal hypergraphic automata. Their subautomata cover all homomorphic images of hypergraphic automata (Theorem 1). Such universal automaton for any hypergraphs H_1 and H_2 is the automaton $\mathrm{Atm}(H_1, H_2) = (H_1, S, H_2, \delta^\circ, \lambda^\circ)$, where S is the input symbol semigroup consisting of all pairs $s = (\varphi, \psi)$ of endomorphisms φ of the hypergraph H_1 and homomorphisms ψ from the hypergraph H_1 to the hypergraph H_2, $\delta^\circ(x, s) = \varphi(x)$ is the transition function and $\lambda^\circ(x, s) = \psi(x)$ is the output function (where x is a vertex of H_1 and $s = (\varphi, \psi)$ is an element of S).

According to our previous study, the universal hypergraphic automata are defined

up to isomorphism by their input symbol semigroups [8]. Additionally, we solved the problem of concrete characterization of universal automata [9]. The main result of our current study is Theorem 2. It shows the important property of input symbol semigroup of universal hypergraphic automaton which allows to construct an isomorphic copy of the original automaton using input symbol semigroup. Such structure is one of the major tools for proving relatively elementary definability [10] of considered universal hypergraphic automata in a class of semigroups in order to analyze interrelation of elementary characteristics of universal hypergraphic automata and their input symbol semigroups.

The main result of the paper was announced at X International Conference "Dicsrete Models in Control Systems Theory".

The authors would like to thank the reviewer for his constructive comments on the paper.

1. Hypergraphic automata

According to A.Bretto [6], a hypergraph is an algebraic system $H = (X, L)$, where X is a nonempty vertex set and L is a family of subsets of the set X called hypergraph edges (or hyperedges). A subset $Y \subseteq X$ is said to be bounded if $Y \subseteq l$ for some $l \in L$, and Y is said to be unbounded otherwise. If hypergraph vertices are incident to some edge, they are called adjacent vertices. The hypergraph is said to be an effective hypergraph if any vertex is incident to some edge of such hypergraph.

Let p be some natural number. The hypergraph H is a hypergraph with p-definable edges if every edge of such hypergraph contains at least $p+1$ vertices and, any p vertices of such hypergraph are incident to no more than one edge.

For example, if we consider planes [7] as hypergraphs with plane points as vertices and plane lines as edges, then any projective plane and any affine plane containing more than 4 points are effective hypergraps with 2-definable edges. Additionally, weak hypergraps studied by A.Molchanov [11] are effective hypergraphs with p-definable edges. Besides, hypergraphs with edges which form partitions of vertex set into equivalence classes containing at least $p + 1$ vertices are also effective hypergraphs with p-definable edges.

In additon to such known examples, there are a lot of non-trivial effective hypergraps with p-definable edges for any natural p.

Example 1. *The hypergraph $H = (X, L)$ with the vertex set $X = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and the edges set $L = \{\{1, 2, 3, 4\}, \{1, 5, 6, 7\}, \{1, 2, 5, 8\}\}$ is an effective hypergraph with 3-definable edges(Fig. 1).*

Let $H_1 = (X, L_X)$, $H_2 = (Y, L_Y)$ be any hypergraphs. A homomorphism from H_1 to H_2 is a mapping φ of the set X to the set Y such that adjacent vertices of the hypergraph H_1 are mapped to adjacent vertices of the hypergraph H_2, i.e. the following condition is satisfied

$$(\forall l \in L_X)(\exists l' \in L_Y)(\varphi(l) \subseteq l').$$

Besides, for any $l \in L_Y$ any mapping $\varphi : X \longrightarrow l$ is a homomorphism from H_1 to H_2.

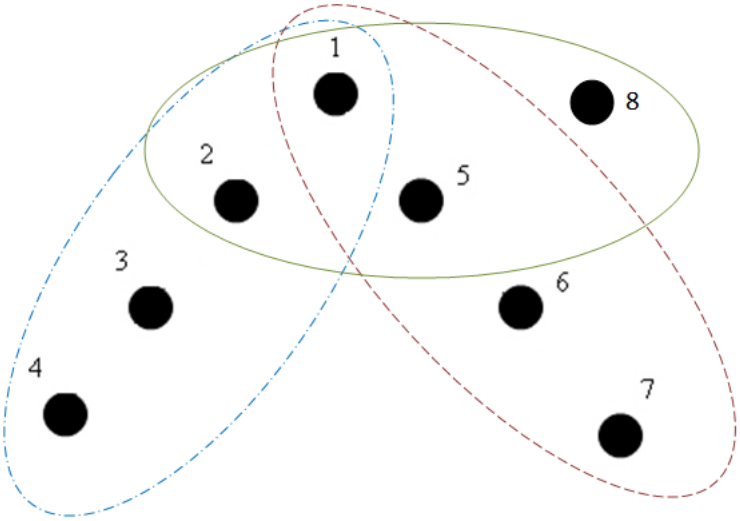


Fig. 1. The effective hypergraph with 3– definable edges H

The set of all homomorphisms from a hypergraph H_1 to a hypergraph H_2 is denoted by $\mathrm{Hom}(H_1, H_2)$. A homomorphism from $H = (X, L)$ to itself is called an endomorphism of H. The set of all endomorphisms of any hypergraph H under the composition operation forms the semigroup $\mathrm{End}\, H$. For hypergraphs $H_1 = (X, L_Y), H_2 = (Y, L_Y)$ by $S(H_1, H_2)$ denote the semigroup $\mathrm{End}\, H_1 \times \mathrm{Hom}(H_1, H_2)$ with binary operation defined by the rule [1]: $(\varphi, \psi)(\varphi_1, \psi_1) = (\varphi\varphi_1, \varphi\psi_1)$ for pairs $(\varphi, \psi), (\varphi_1, \psi_1) \in \mathrm{End}\, H_1 \times \mathrm{Hom}(H_1, H_2)$.

From the algebraic point of view an effective hypergraph with p-definable edges $H = (X, L)$ is an algebraic system $H = (X, L, \rho)$ consisting of two sets X, L and a binary relation $\rho \subset X \times L$, which is defined by the formula $(x, l) \in \rho \iff x \in l$ (where $x \in X, l \in L$) and fulfills the conditions:

$$(\Gamma_1) \quad (\forall x \in X)\,(\exists l \in L)\,((x, l) \in \rho),$$

$$(\Gamma_2) \quad (\forall l \in L)\,(\exists x_1, x_2, ..., x_{p+1} \in X)(\bigwedge_{1 \le i \ne j \le p+1} x_i \ne x_j \wedge \bigwedge_{1 \le i \le p+1} (x_i, l) \in \rho),$$

$$(\Gamma_3)(\forall x_1, x_2, ..., x_p \in X)(\bigwedge_{1 \le i \ne j \le p} x_i \ne x_j \Rightarrow (\forall l, r)(\bigwedge_{1 \le i \le p} ((x_i, l) \in \rho \wedge (x_i, r) \in \rho)) \Rightarrow$$

$$\Rightarrow r = l)).$$

An isomorphism of such system $H_1 = (X, L_X, \rho)$, $H_2 = (Y, L_Y, \rho')$ is an ordered pair $\pi = (\varphi, \psi)$ of bijections $\varphi : X \to Y$, $\psi : L_X \to L_Y$ preserving system relations, i.e. for any $x \in X, l \in L_X$ the condition $(x, l) \in \rho \iff (\varphi(x), \psi(l)) \in \rho'$ holds.

An automaton $A = (X, S, Y, \delta, \lambda)$ is a hypergraphic automaton if state set X and output symbol set Y are such hypergraphs $H_1 = (X, L_X)$ and $H_2 = (Y, L_Y)$ respectively that for every fixed input symbol $s \in S$ the transformation $\delta_s : X \longrightarrow X$ is an endomorphism of H_1 and the mapping $\lambda_s : X \longrightarrow Y$ is a homomorphism from H_1 to H_2. Such automata is also denoted as $A = (H_1, S, H_2, \delta, \lambda)$.

An input symbol $a \in S$ of automata $A = (X, S, Y, \delta, \lambda)$ is called autonomous if its action is independent of the automaton state, i.e. there is such automaton state, denoted by a_1, and such output automaton symbol, denoted by a_2, that $\delta(x, a) = a_1$, $\lambda(x, a) = a_2$ for all automaton states $x \in X$.

Let $H_1 = (X, L_X), H_2, H_1', H_2'$ be arbitrary hypergraphs, $A = (H_1, S, H_2, \delta, \lambda)$, $A' = (H_1', S', H_2', \delta', \lambda')$ be hypergraphic automata. A homomorphism from A to A' is an ordered triple $\theta = (\pi_1, \gamma, \pi_2)$ of hypergraph homomorphisms $\pi_1 = (\varphi_1, \psi_1) : H_1 \to H_1'$, $\pi_2 = (\varphi_2, \psi_2) : H_2 \to H_2'$ and a semigroup homomorphism $\gamma : S \to S'$, preserving transition functions and output functions of the automata, i.e. the formulas

$$\varphi_1\left(\delta(x, s)\right) = \delta'\left(\varphi_1(x), \gamma(s)\right), \quad \varphi_2\left(\lambda(x, s)\right) = \lambda'\left(\varphi_1(x), \gamma(s)\right)$$

hold for any $x \in X, s \in S$. If π_1, π_2, γ are isomorphisms, then θ is called an isomorphism of hypergraphic automata A and A'.

The important example of hypergraphic automaton is an algebraic system $\mathrm{Atm}(H_1, H_2) = (H_1, S(H_1, H_2), H_2, \delta^\circ, \lambda^\circ)$, where $H_1 = (X, L_X)$, $H_2 = (Y, L_Y)$ are some hypergraphs and for any $x \in X$, $(\varphi, \psi) \in S(H_1, H_2)$ the conditions hold: $\delta^\circ(x, (\varphi, \psi)) = \varphi(x)$, $\lambda^\circ(x, (\varphi, \psi)) = \psi(x)$.

For a set X, let Δ_X denote the identity transformation of X.

Theorem 1. *For any hypergraphic automaton $A = (H_1, S, H_2, \delta, \lambda)$ with state hypergraph $H_1 = (X, L_X)$ and output symbol hypergraph $H_2 = (Y, L_Y)$ there is such homomorphism $\pi : S \longrightarrow S(H_1, H_2)$ that ordered triple $\gamma = (\Delta_X, \pi, \Delta_Y)$ is a homomorphism from the automaton A to the automaton* $\mathrm{Atm}(H_1, H_2)$.

Proof. By definition of hypergraphic automaton $A = (H_1, S, H_2, \delta, \lambda)$, for any $s \in S$ the transformation $\delta_s : X \longrightarrow X$ is an endomorphism of the hypergraph H_1 and the mapping $\lambda_s : X \longrightarrow Y$ is a homomorphism from the hypergraph H_1 to the hypergraph H_2. Thus, $(\delta_s, \lambda_s) \in S(H_1, H_2)$ and we can define a mapping $\pi : S \longrightarrow S(H_1, H_2)$ by the following rule: $\pi(s) = (\delta_s, \lambda_s)$ for every $s \in S$. In accordance with the conditions of interrelation between the input symbol semigroup S and the transition function δ and the output function λ of the automaton A, for every $s, t \in S$ the equalities hold:

$$\pi(s \cdot t) = (\delta_{s \cdot t}, \lambda_{s \cdot t}) = (\delta_s \delta_t, \delta_s \lambda_t) = (\delta_s, \lambda_s)(\delta_t, \lambda_t) = \pi(s)\pi(t).$$

Hence, the mapping π is a homomorphism from the semigroup S to the semigroup $S(H_1, H_2)$. We prove that ordered triple $\gamma = (\Delta_X, \pi, \Delta_Y)$ is a homomorphism from the automaton A to the automaton $\mathrm{Atm}(H_1, H_2)$. It is easy to show that for any state $x \in X$ of the automaton A and any input symbol $s \in S$ of the automaton the equalities hold:

$$\Delta_X(\delta(x, s)) = \delta(x, s) = \delta_s(x) = \delta^\circ(x, \pi(s)) = \delta^\circ(\Delta_X(x), \pi(s)),$$

$$\Delta_Y(\lambda(x, s)) = \lambda(x, s) = \lambda_s(x) = \lambda^\circ(x, \pi(s)) = \lambda^\circ(\Delta_X(x), \pi(s)).$$

Thus, γ is a homomorphism from A to $\mathrm{Atm}(H_1, H_2)$. □

Hence, the automaton $\mathrm{Atm}(H_1, H_2)$ is a universally attracting object [1] of a category of hypergraphic automata with the state hypergraph H_1 and the output symbol hypergraph H_2. Therefore, $\mathrm{Atm}(H_1, H_2)$ is called a universal hypergraphic automaton for the hypergraphs H_1, H_2.

2. Preliminaries

Now we consider the universal hypergraphic automaton $A = \mathrm{Atm}(H_1, H_2)$ for some effective hypergraphs with p-definable edges $H_1 = (X_1, L_1)$ and $H_2 = (X_2, L_2)$. Let C be the set of all autonomous input symbols of the automaton A. Define canonical relations for such automaton:

1) the binary relation ε_1 on C, consisting of such ordered pairs (a, b) of autonomous input symbols $a, b \in C$, which transform states of the automaton A identically, i.e. $(a, b) \in \varepsilon_1 \iff a_1 = b_1$;

2) the binary relation ε_2 on C, consisting of such ordered pairs (a, b) of autonomous input symbols $a, b \in C$, which generate the same output symbols of the automaton A, i.e. $(a, b) \in \varepsilon_2 \iff a_2 = b_2$;

3) the binary relation η_i on C^p, consisting of such ordered pairs (α, β) of elements $\alpha = (a^1, a^2, ..., a^p)$ and $\beta = (b^1, b^2, ..., b^p)$, $a^1, a^2, ..., a^p, b^1, b^2, ..., b^p \in C$, that for every $i = 1, 2$ they map states of A to the bounded set $\{a_i^1, a_i^2, ..., a_i^p, b_i^1, b_i^2, ..., b_i^p\}$ of H_i, i.e.

$$(\alpha, \beta) \in \eta_i \iff \{a_i^1, a_i^2, ..., a_i^p, b_i^1, b_i^2, ..., b_i^p\} \text{ is a bounded set of } H_i \ (i = 1, 2).$$

Let $D_i, i = 1, 2$ denote the set consisting of ordered p-tuples of autonomous input symbols $x^1, x^2, ..., x^p$ such that: $x_i^k \neq x_i^j$ for all $1 \leq k < j \leq p$ and the set $\{x_i^1, x_i^2, ..., x_i^p\}$ is a bounded set of H_i.

Lemma 1. *Let $H_1 = (X_1, L_1)$, $H_2 = (X_2, L_2)$ be effective hypergraps with p-definable edges. Then the canonical relations of the universal hypergraphic automaton $A = \mathrm{Atm}(H_1, H_2)$ satisfy the conditions:*

1) for any state (output symbol, respectively) x of the automaton A there is such autonomous input symbol of the automaton denoted by $\widetilde{x}$ that the automaton A jumps from any state to the state x (outputs symbol x for any state, respectively) due to $\widetilde{x}$, i.e. the condition $\widetilde{x}_1 = x$ holds ($\widetilde{x}_2 = x$, respectively);

2) for each $i = 1, 2$ the relation ε_i is an equivalence relation on the set C and the mapping $\varphi_i : X_i \to C/\varepsilon_i$ defined for $x \in X_i$ by the formula $\varphi_i(x) = \varepsilon_i(\widetilde{x})$ is a bijection from X_i to the factor set C/ε_i;

3) for each $i = 1, 2$ the restriction of η_i to the set D_i is a equivalence relation such that the mapping $\psi_i : L_i \to D_i/\eta_i$ defined for $l \in L_i$ by the formula $\psi_i(l) = \eta_i(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p})$ for arbitrary pairwise distinct vertices $x^1, x^2, ..., x^p \in l$ is a bijection from L_i to the factor set D_i/η_i.

Proof. Because the hypergraphs H_1, H_2 are effective, i.e. they satisfy the condition Γ_1, for any state x and any output symbol y of A constant mappings $\varphi : X_1 \to \{x\}$ and $\psi : X_1 \to \{y\}$ are the endomorphism of H_1 and the homomorphism from H_1 to H_2, respectively. Then the pair of mappings $a = (\varphi, \psi)$ is autonomous input symbol of A satisfying the conditions: $a_1 = x, a_2 = y$. Thus, the statement 1) of the lemma is correct.

The statement 2) of the lemma is obviously true.

Consider the restriction of the relation η_1 to the set D_1. Let $\alpha = (a^1, a^2, ..., a^p)$ be an arbitrary element of D_1. According to the definition of the set D_1, $\alpha \in C^p$ and $\{a_1^1, a_1^2, ..., a_1^p\}$ is a bounded set of H_1. Thus, by definition of η_1, the pair $(\alpha, \alpha) \in \eta_1$. Hence, η_1 is a reflexive relation.

For any $\alpha, \beta \in D_1$, where $\alpha = (a^1, a^2, ..., a^p)$, $\beta = (b^1, b^2, ..., b^p)$ for some $a^1, a^2, ..., a^p, b^1, b^2, ..., b^p \in C$, the condition $(\alpha, \beta) \in \eta_1$ means that $\{a_1^1, a_1^2, ..., a_1^p, b_1^1, b_1^2, ..., b_1^p\}$ is a bounded set of H_1. Thus, $\{b_1^1, b_1^2, ..., b_1^p, a_1^1, a_1^2, ..., a_1^p\}$ is also a bounded set of H_1, i.e. $(\beta, \alpha) \in \eta_1$. Hence, η_1 is a symmetric relation.

To prove transitivity of the relation we consider any $\alpha, \beta, \gamma \in D_1$ where $\alpha = (a^1, a^2, ..., a^p)$, $\beta = (b^1, b^2, ..., b^p)$, $\gamma = (c^1, c^2, ..., c^p)$ for some $a^1, a^2, ..., a^p, b^1, b^2, ..., b^p, c^1, c^2, ..., c^p \in C$ satisfying the conditions $(\alpha, \beta), (\beta, \gamma) \in \eta_1$. Thus, the sets $\{a_1^1, a_1^2, ..., a_1^p, b_1^1, b_1^2, ..., b_1^p\}$, $\{b_1^1, b_1^2, ..., b_1^p, c_1^1, c_1^2, ..., c_1^p\}$ are bounded sets of H_1, i.e. there are such edges $l_1, l_2 \in L_1$ that $\{a_1^1, a_1^2, ..., a_1^p, b_1^1, b_1^2, ..., b_1^p\} \subseteq l_1$, $\{b_1^1, b_1^2, ..., b_1^p, c_1^1, c_1^2, ..., c_1^p\} \subseteq l_2$. According to the definition of the set D_1, $a_1^k \neq a_1^j$, $b_1^k \neq b_1^j$, $c_1^k \neq c_1^j$ for all $1 \leq k < j \leq p$. By definition of a hypergraph with p-definable edges, any edge of such hypergraph is uniquely determined by any its distinct p vertices $b_1^1, b_1^2, ..., b_1^p$, i.e. it satisfies the condition Γ_3. Hence, $l_1 = l_2 = l$. Thus, the set $\{a_1^1, a_1^2, ..., a_1^p, c_1^1, c_1^2, ..., c_1^p\} \subseteq l$ is a bounded set of the hypergraph H_1. According to the definition of the relation η_1, we have $(\alpha, \gamma) \in \eta_1$. Hence, the relation η_1 is a transitive relation. Therefore, η_1 is an equivalence relation on D_1, which defines the factor set D_1/η_1.

By definition of a hypergraph with p-definable edges, for each edge $l \in L_1$ there are p distinct vertices $x^1, x^2, ..., x^p \in X_1$ such that $x^1, x^2, ..., x^p \in l$, i.e. the set $\{x^1, x^2, ..., x^p\}$ is a bounded set of H_1. As shown above, there are such input symbols $\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p}$ of the automaton A that $\widetilde{x_1^1} = x^1, \widetilde{x_1^2} = x^2, ..., \widetilde{x_1^p} = x^p$. As $x^k \neq x^j$, $1 \leq k < j \leq p$, the tuple $\alpha = \left(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p}\right)$ is contained in the set D_1 and defines an equivalence class $\eta_1(\alpha)$.

Denote $\psi_1(l) = \eta_1(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p})$. As for any p distinct vertices $y^1, y^2, ..., y^p \in l$, autonomous input symbols $\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p}, \widetilde{y^1}, \widetilde{y^2}, ..., \widetilde{y^p}$ define adjacent vertices $\widetilde{x_1^1} = x^1, \widetilde{x_1^2} = x^2, ..., \widetilde{x_1^p} = x^p, \widetilde{y_1^1} = y^1, \widetilde{y_1^2} = y^2, ..., \widetilde{y_1^p} = y^p$ of H_1 (the set $\{x^1, x^2, ..., x^p, y^1, y^2, ..., y^p\}$ is a bounded set of H_1), the condition $\left(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p}\right) \equiv \left(\widetilde{y^1}, \widetilde{y^2}, ..., \widetilde{y^p}\right) (\eta_1)$ holds. Thus, the definition of $\psi_1(l)$ is correct.

Prove that ψ_1 is a bijection from L_1 to the factor set D_1/η_1. For any equivalence class $\eta_1(a^1, a^2, ..., a^p)$ defined by an ordered set $(a^1, a^2, ..., a^p) \in D_1$, we have $a^1, a^2, ..., a^p \in C$, $a_1^k \neq a_1^j$ for all $1 \leq k < j \leq p$, and the set $\{a_1^1, a_1^2, ..., a_1^p\}$ is a bounded set of H_1. Thus, in H_1 there is such edge $l \in L_1$ that $\{a_1^1, a_1^2, ..., a_1^p\} \subseteq l$. Hence, by definition $\psi_1(l) = \eta_1(a^1, a^2, ..., a^p)$, i.e. the mapping ψ_1 is a surjection from L_1 to the factor set D_1/η_1.

On the other hand, according to the definition of the hypergraph with p-definable

edges H_1, for any edges $l, r \in L_1$ satisfying $l \neq r$, there are such vertices $x^1, x^2, ..., x^p, y^1, y^2, ..., y^p \in X_1$ that $x^k \neq x^j$, $y^k \neq y^j$, $1 \leq k < j \leq p$, $x^1, x^2, ..., x^p \in l$, $y^1, y^2, ..., y^p \in r$, and at least one of vertices $y^1, y^2, ..., y^p$ is not contained in l. Then the ordered sets of autonomous input symbols $\alpha = \left(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p}\right)$, $\beta = \left(\widetilde{y^1}, \widetilde{y^2}, ..., \widetilde{y^p}\right)$ satisfy the conditions $\alpha, \beta \in D_1$ and $\alpha \not\equiv \beta(\eta_1)$ because vertices $\widetilde{x_1^1} = x^1, \widetilde{x_1^2} = x^2, ..., \widetilde{x_1^p} = x^p$, $\widetilde{y_1^1} = y^1, \widetilde{y_1^2} = y^2, ..., \widetilde{y_1^p} = y^p$ can not belong to the same edge by definiton of the hypergraph with p-definable edges H_1 (the property Γ_3). Hence, the conditions hold: $\psi_1(l) = \eta_1\left(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p}\right) = \eta_1(\alpha)$, $\psi_1(r) = \eta_1\left(\widetilde{y^1}, \widetilde{y^2}, ..., \widetilde{y^p}\right) = \eta_1(\beta)$, $\psi_1(l) \neq \psi_1(r)$. Therefore, ψ_1 is one-to-one mapping. Thus, ψ_1 is a bijection from the set L_1 to the factor set D_1/η_1.

It is easy to prove in a similar way that the mapping ψ_2 is a bijection from the set L_2 to the factor set D_2/η_2. Henceforth, the statement 3) of the lemma is true. □

3. Main result

Let $A = \mathrm{Atm}(H_1, H_2)$ be a universal hypergraphic automaton for some effective hypergraphs with p-definable edges $H_1 = (X_1, L_1)$ and $H_2 = (X_2, L_2)$. We introduce the following concepts using the automaton canonical relations:

1) for every $i = 1, 2$ define an algebraic system $\overline{H}_i = \left(\overline{X}_i, \overline{L}_i, \overline{\rho}_i\right)$ with two carrier sets $\overline{X}_i = C/\varepsilon_i$, $\overline{L}_i = D_i/\eta_i$ and a binary relation $\overline{\rho}_i \subset \overline{X}_i \times \overline{L}_i$, which is defined for elements $a, a^1, a^2, ..., a^p \in C$, $a^k \not\equiv a^j(\varepsilon_i), 1 \leq k < j \leq p$ by the formula:

$(\varepsilon_i(a), \eta_i(a^1, a^2, ..., a^p)) \in \overline{\rho}_i \iff \{a_i, a_i^1, a_i^2, ..., a_i^p\}$ is a bounded set of H_i;

2) define two mappings $\overline{\delta} : \overline{X}_1 \times S \to \overline{X}_1$, $\overline{\lambda} : \overline{X}_1 \times S \to \overline{X}_2$ by the formulas for $a \in C, s \in S$:

$$\overline{\delta}\,(\varepsilon_1(a), s) = \varepsilon_1(a \cdot s), \quad \overline{\lambda}\,(\varepsilon_1(a), s) = \varepsilon_2(a \cdot s).$$

Theorem 2. *Let $A = \mathrm{Atm}(H_1, H_2)$ be a universal hypergraphic automaton for some effective hypergraphs with p-definable edges $H_1 = (X_1, L_1)$ and $H_2 = (X_2, L_2)$. Then the following statements are true:*

1) for every $i = 1, 2$ the hypergraph H_i is isomorphic to the algebraic system $\overline{H}_i = \left(\overline{X}_i, \overline{L}_i, \overline{\rho}_i\right)$;

2) the automaton $A = \mathrm{Atm}(H_1, H_2)$ is isomorphic to a hypergraphic automaton $\overline{A} = \left(\overline{H}_1, S, \overline{H}_2, \overline{\delta}, \overline{\lambda}\right)$ with the state hypergraph $\overline{H}_1$, the input symbol semigroup $S = \mathrm{Inp}(A)$, the output symbol hypergraph $\overline{H}_2$, the transition function $\overline{\delta} : \overline{X}_1 \times S \to \overline{X}_1$, and the output function $\overline{\lambda} : \overline{X}_1 \times S \to \overline{X}_2$.

Proof. Consider an algebraic system $\overline{H}_1 = \left(\overline{X}_1, \overline{L}_1, \rho_1\right)$ with two basic sets $\overline{X}_1 = C/\varepsilon_1$, $\overline{L}_1 = D_1/\eta_1$ and the binary relation $\rho_1 \subset \overline{X}_1 \times \overline{L}_1$, which is defined for elements $a, a^1, a^2, ..., a^p \in C$, $a^k \not\equiv a^j(\varepsilon_i)$, $1 \leq k < j \leq p$ by the formula :

$(\varepsilon_1(a), \eta_1(a^1, a^2, ..., a^p)) \in \overline{\rho}_1 \iff \{a_1, a_1^1, a_1^2, ..., a_1^p\}$ is a bounded set of H_1.

According to the statement 2) of Lemma 1, the mapping $\varphi_i : X_1 \to \overline{X}_1$ defined for $x \in X_1$ by the formula $\varphi_1(x) = \varepsilon_1(\widetilde{x})$ is a bijection from the set X_1 to the set $\overline{X}_1$.

In accordance with the statement 3) of Lemma 1, the mapping $\psi_1 : L_1 \to \overline{L}_1$ defined for element $l \in L_1$ by the formula $\psi_1(l) = \eta_1(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p})$ for any distinct vertices $x^1, x^2, ..., x^p \in l$ is a bijection from the set L_1 to the set $\overline{L}_1$.

Let a vertex $x \in X_1$ is incident to an edge $l \in L_1$. Then for the hypergraph with p-definable edges H_1 by the property Γ_2, there are at least p distinct vertices $x^1, x^2, ..., x^p \in X_1$ such that $x^1, x^2, ..., x^p \in l$. It was already proved that in the automaton A there are such autonomous input symbols $\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p}, \widetilde{x}$ that $\widetilde{x}_1^1 = x^1, \widetilde{x}_1^2 = x^2, ..., \widetilde{x}_1^p = x^p, \widetilde{x}_1 = x$. Then $\varphi_1(x) = \varepsilon_1(\widetilde{x})$, $\psi_1(l) = \eta_1\left(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p}\right)$, and $\{\widetilde{x}_1^1, \widetilde{x}_1^2, ..., \widetilde{x}_1^p, \widetilde{x}_1\} \subseteq l$. Thus, by definition of $\overline{\rho}_1$, the condition $(\varphi_1(x), \psi_1(l)) \in \overline{\rho}_1$ holds.

On the other hand, let $(\varphi_1(x), \psi_1(l)) \in \overline{\rho}_1$ for some elements $x \in X_1$, $l \in L_1$. Then $\varphi_1(x) = \varepsilon_1(\widetilde{x})$, $\psi_1(l) = \eta_1(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p})$ for some distinct vertices $x^1, x^2, ..., x^p \in l$ and $\left(\varepsilon_1(\widetilde{x}), \eta_1(\widetilde{x^1}, \widetilde{x^2}, ..., \widetilde{x^p})\right) \in \overline{\rho}_1$. According to the definition of the relation $\overline{\rho}_1$, this condition means that vertices $\widetilde{x}_1^1 = x^1, \widetilde{x}_1^2 = x^2, ..., \widetilde{x}_1^p = x^p, \widetilde{x}_1 = x$ belongs to the same edge r of the hypergraph H_1. In accordance with definition of an effective hypergraph with p-definable edges, any edge is uniquely determined by any p vertices $x^1, x^2, ..., x^p$ (the condition Γ_3). Thus, $l = r$. Therefore, $x \in l$. Hence, $\pi_1 = (\varphi_1, \psi_1)$ is an isomorphism from the hypergraph H_1 to the algebraic system $\overline{H}_1$.

It is easy to prove in a similar way that the ordered pair of mappings $\pi_2 = (\varphi_2, \psi_2)$ is an isomorphism from the hypergraph H_2 to the algebraic system $\overline{H}_2$. Thus, the statement 1) of the theorem is true.

It follows from 1), that the algebraic systems $\overline{H}_1 = \left(\overline{X}_1, \overline{L}_1, \overline{\rho}_1\right)$, $\overline{H}_2 = \left(\overline{X}_2, \overline{L}_2, \overline{\rho}_2\right)$ are effective hypergraphs with p-definable edges and the algebraic system $\overline{A} = \left(\overline{H}_1, S, \overline{H}_2, \overline{\delta}, \overline{\lambda}\right)$ is a hypergraphic automaton for the hypergraphs H_1, H_2. We denote the ordered triple (π_1, Δ_S, π_2) by θ and prove that θ is an isomorphism from the universal hypergraphic automaton $A = \mathrm{Atm}(H_1, H_2)$ to the hypergraphic automaton $\overline{A}$. It remains to prove that the triple θ preserves transiton functions and output functions of the automata, i.e. for any $x \in X_1, s \in S$ the conditions hold:

$$\varphi_1\left(\delta^\circ(x, s)\right) = \overline{\delta}\left(\varphi_1(x), s\right), \quad \varphi_2\left(\lambda^\circ(x, s)\right) = \overline{\lambda}\left(\varphi_1(x), s\right).$$

For any automaton state $x \in X_1$ the value $\widetilde{x}$ is an autonomous input symbol such that the automaton jumps to the state x, i.e. the condition $\widetilde{x}_1 = x$ holds. It is easy to see that for any input symbol $s \in S$ the composition $\widetilde{x} \cdot s$ is an autonomous input symbol of the automaton A and the automaton jumps to the state $\delta^\circ(x, s)$ as well as generates the symbol $\lambda^\circ(x, s)$ depending on $\widetilde{x} \cdot s$, i.e. the conditions hold: $(\widetilde{x} \cdot s)_1 = \delta^\circ(x, s)$, $(\widetilde{x} \cdot s)_2 = \lambda^\circ(x, s)$. Thus, by definition of δ°, $\widetilde{\delta^\circ(x, s)} = \widetilde{x} \cdot s$ and the equalities hold:

$$\varphi_1\left(\delta^\circ(x, s)\right) = \varepsilon_1\left(\widetilde{\delta^\circ(x, s)}\right) = \varepsilon_1\left(\widetilde{x} \cdot s\right) = \overline{\delta}\left(\varepsilon_1\left(\widetilde{x}\right), s\right) = \overline{\delta}\left(\varphi_1(x), s\right).$$

Similarly, by definition of λ°, $\widetilde{\lambda^\circ(x, s)} = \widetilde{x} \cdot s$ and the equalities hold:

$$\varphi_2\left(\lambda^\circ(x, s)\right) = \varepsilon_2\left(\widetilde{\lambda^\circ(x, s)}\right) = \varepsilon_2\left(\widetilde{x} \cdot s\right) = \overline{\lambda}\left(\varepsilon_1\left(\widetilde{x}\right), s\right) = \overline{\lambda}\left(\varphi_1(x), s\right).$$

Hence, $\theta = (\pi_1, \Delta_S, \pi_2)$ is an isomorphism from the universal hypergraphic automaton $A = \mathrm{Atm}(H_1, H_2)$ to the hypergraphic automaton $\overline{A}$. □

4. Conclusions

The main result of the paper allows us to represent a universal hypergraphic automaton for effective hypergraphs with p-definable edges as an algebraic structure canonically constructed in the input symbol semigroup of the automaton. This representation gives us an effective tool for studying a correlation between of properties of universal hypergraphic automata and their input symbol semigroups. The major tools of the representation of a universal hypergraphic automaton in his input symbol semigroup are the canonical relations of the automaton. We plan to prove that these relations are defined by formulas of the first order logic in the input symbols semigroups of the automata.

Based on our approach, we will prove the relatively elementary definability [10] of the class of considered universal hypergraphic automata in the class of all semigroups. It will allow us to investigate the abstract representation problem for universal hypergraphic automata, the elementary definability problem of universal hypergraphic automata by their input symbol semigroup, the algorithmic solvability problem of elementary theories of universal hypergraphic automata, and others.

References

[1] Plotkin B. I., Geenglaz L. Ja., Gvaramija A. A., *Algebraic structures in automata and databases theory*, World Scientific, Singapore, 1992.

[2] Simovici D., "On the theory of reduction of semilatticial automata", *Scientific Annals of the Alexandru Ioan Cuza University of Iasi (New Series)*, **22**:1 (1976), 107–110.

[3] Gecseg F., "O proizvedeniyakh uporyadochennykh avtomatov. I", *Acta Sci. Math.*, **24**:3–4 (1963), 244–250 (in Russian).

[4] Gecseg F., "O proizvedeniyakh uporyadochennykh avtomatov. II", *Acta Sci. Math.*, **25**:1–2 (1964), 124–128 (in Russian).

[5] Eilenberg S., *Automata, languages and machines. Volume B*, Academic Press, New York, San Francisco, London, 1976.

[6] Bretto A., *Hypergraph theory. An Introduction*, Springer, Cham, 2013.

[7] Hartshorne R., *Foundations of Projective Geometry*, Ishi Press, New York, 2009.

[8] Molchanov V. A., Khvorostukhina E. V., "Ob abstraktnoy opredelyaemosti universalnykh gipergraficheskikh avtomatov polugruppami ikh vkhodnykh signalov", *Issledovaniya po algebre, teorii chisel, funktsionalnomu analizu i smezhnym voprosam*, 2016, № 8, 67–69 (in Russian).

[9] Molchanov V. A., Khvorostukhina E. V., "On problem of concrete characterization of universal automata", *Lobachevskii Journal of Mathematics*, **38**:4 (2017), 664–669.

[10] Ershov Yu. L., *Problemy razreshimosti i konstruktivnye modeli*, Nauka, Moscow, 1980 (in Russian).

[11] Molchanov A. V., "Endomorphism semigroups of weak p-hypergraphs", *Russian Mathematics (Izvestiya VUZ. Matematika)*, **44**:3 (2000), 77–80.

Хворостухина Е. В., Молчанов В. А., "Представление универсальных гиперграфических автоматов автономными выходными сигналами", *Моделирование и анализ информационных систем*, **25**:5 (2018), 561–571.

DOI: 10.18255/1818-1015-2018-5-561-571

Аннотация. Гиперграфическими автоматами называются автоматы, у которых множества состояний и выходных символов наделены структурами гиперграфов, сохраняющимися функциями переходов и выходными функциями. Универсальные притягивающие объекты в категории таких автоматов представляются автоматами $\mathrm{Atm}(H_1, H_2)$ с гиперграфом состояний H_1, гиперграфом выходных символов H_2 и полугруппой входных символов $S = \mathrm{End}\, H_1 \times \mathrm{Hom}(H_1, H_2)$, которые называются универсальными гиперграфическими автоматами. Для такого автомата $\mathrm{Atm}(H_1, H_2)$ полугруппа входных символов S является производной алгеброй отображений, свойства которой взаимосвязаны со свойствами алгебраической структуры данного автомата. Это позволяет изучать универсальные гиперграфические автоматы с помощью исследования их полугрупп входных символов. В настоящей работе рассматривается проблема представления универсальных гиперграфических автоматов в их полугруппах входных сигналов: описывается представление универсального гиперграфического автомата в виде многосортной алгебраической системы, канонически построенной из автономных входных сигналов этого автомата. Эта конструкция является одним из инструментов доказательства относительно элементарной определимости рассматриваемых автоматов в классе полугрупп, которая позволяет проанализировать взаимосвязь элементарных свойств этих автоматов и их полугрупп входных сигналов. Основной результат работы дает решение этой задачи для универсальных гиперграфических автоматов над эффективными гиперграфами с p-определимыми ребрами. Это достаточно широкий и весьма важный класс автоматов, так как он содержит, в частности, автоматы, у которых гиперграфы состояний и выходных символов являются плоскостями (например, проективными или аффинными) или разбиениями на классы нетривиальных эквивалентностей. Статья публикуется в авторской редакции.

Ключевые слова: автомат, полугруппа, гиперграф, входной сигнал

Об авторах:
Хворостухина Екатерина Владимировна, orcid.org/0000-0002-2775-5732, канд. физ.-мат. наук, доцент,
Саратовский государственный технический университет им. Гагарина Ю.А.,
ул. Политехническая, 77, г. Саратов, 410054 Россия, e-mail: khvorostukhina85@gmail.com

Молчанов Владимир Александрович, orcid.org/0000-0001-6509-3090, докт. физ.-мат. наук, профессор,
Саратовский национальный исследовательский государственный университет им. Н.Г. Чернышевского,
ул. Астраханская, 83, г. Саратов, 410012 Россия, e-mail: molchanovva@mail.ru

Моделирование и анализ информационных систем. Т. 25, №5 (2018), с. 572–583
Modeling and Analysis of Information Systems. Vol. 25, No 5 (2018), pp. 572–583

Динамика нейронных сетей
Neural Network Dynamics

DOI: 10.18255/1818-1015-2018-5-572-583

УДК 517.9

Неупорядоченные колебания в нейросети из трех осцилляторов с запаздывающей вещательной связью

Глызин С. Д., Марушкина Е. А.

получена 3 сентября 2018

Аннотация. Рассматривается модель нейронной ассоциации из трех импульсных нейронов с вещательной запаздывающей связью между ними. Учитывая, что связь вещательная, в системе отщепляется уравнение, соответствующее одному из осцилляторов. Два оставшихся импульсных нейрона взаимодействуют друг с другом, и, кроме того, имеется периодическое внешнее воздействие, определяемое вещательным нейроном. В этих условиях, при значениях параметров, близких к критическим, на устойчивом инвариантном интегральном многообразии построена нормальная форма данной системы. Эта нормальная форма сводится к четырехмерной системе, две переменных которой отвечают за амплитуды колебаний осцилляторов, а две другие определяются разностью фазовых переменных этих осцилляторов с фазовой переменной вещательного осциллятора. Полученная нормальная форма имеет инвариантное многообразие, на котором амплитудные и фазовые переменные осцилляторов совпадают. Описана динамика задачи на этом многообразии. Важный результат удалось получить на основе численного анализа нормальной формы. Оказалось, что при ослаблении связи между осцилляторами могут возникать периодические и хаотические колебательные решения. Более того, был обнаружен каскад бифуркаций, связанный с однотипными фазовыми перестройками, в котором поочередно самосимметричный устойчивый цикл теряет симметрию с возникновением двух симметричных друг другу циклов; с каждым из этих циклов происходит каскад бифуркаций удвоения с появлением симметричных хаотических режимов. Эти симметричные хаотические режимы при дальнейшем уменьшении параметра связи объединяются в самосимметричный, который затем перестраивается в самосимметричный цикл более сложного вида по сравнению с полученным на предыдущем шаге. Далее весь процесс повторяется. Для изучения хаотических аттракторов системы вычислялись ляпуновские показатели.

Ключевые слова: осциллятор, нейросеть, вещательная связь, запаздывание, устойчивость, бифуркация, хаотический аттрактор

Для цитирования: Глызин С. Д., Марушкина Е. А., "Неупорядоченные колебания в нейросети из трех осцилляторов с запаздывающей вещательной связью", *Моделирование и анализ информационных систем*, **25**:5 (2018), 572–583.

Об авторах: Глызин Сергей Дмитриевич, orcid.org/0000-0002-6403-4061, д-р физ.-мат. наук, проф., зав. кафедрой компьютерных сетей, Ярославский государственный университет им. П.Г. Демидова, ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: glyzin@uniyar.ac.ru

Марушкина Елена Александровна, orcid.org/0000-0001-9183-6484, канд. физ.-мат. наук, науч. сотр., Ярославский государственный университет им. П.Г. Демидова, Лаборатория дискретной и вычислительной геометрии им. Б. Н. Делоне, ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: marushkina-ea@yandex.ru

Благодарности:
Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 16-31-60039 мол_а_дк.

1. Постановка задачи

Проблема предсказания сложного коллективного поведения ассоциаций связанных осцилляторов различной природы является одной из актуальных задач современной нейродинамики и популяционной биологии. Сети, составленные из нескольких различным способом взаимодействующих осцилляторов, представляют собой универсальный модельный объект, находящий применение в различных областях науки и техники. В работах [1–3] построен полный набор сетей из трех элементов с различной структурой связи между ними. Среди них сеть с вещательным взаимодействием. В этой сети один из осцилляторов не зависит от двух других, но воздействует на них. Эти осцилляторы в свою очередь взаимодействуют друг с другом и получают сигнал от первого.

Математические модели взаимодействия осцилляторов предлагались в большом числе статей [4–8]. В частности, в работах [8, 9] в качестве модельной задачи для описания электрического взаимодействия цепочки импульсных нейронов была предложена следующая система дифференциально-разностных уравнений:

$$\dot{U}_j = \lambda\big[f(U_j(t-1)) + g(U_j(t-\theta))\big]U_j + D(U_{j-1} + U_{j+1} - 2U_j), \quad j = 1, \dots, n, \qquad (1)$$

где $U_0 \equiv U_1$, $U_{n+1} \equiv U_n$ в случае цепочки и $U_0 \equiv U_n$, $U_{n+1} \equiv U_1$ в случае кольца осцилляторов. Функции $U_j(t)$ моделируют мембранные потенциалы соответствующих нейронов, $\lambda > 0$ — параметр, отвечающий за скорость протекания процессов в системе, параметр $D > 0$ определяет силу диффузионной связи между осцилляторами, достаточно гладкие функции $f(U(t-1))$, $g(U(t-\theta))$, $\theta > 0$ моделируют ионную проводимость мембраны нервной клетки для разных типов ионов. Функции $f(U)$ и $g(U)$ обладают следующими предельными свойствами: $f(0) = 1$, $g(0) = 0$ и $f(u) \to -a$ $(a > 0)$, $g(u) \to 1$ при $u \to +\infty$. Рассмотрим систему (1) в случае трех $(n = 3)$ осцилляторов и будем считать, что они связаны между собой вещательным образом:

$$\begin{aligned}
\dot{U}_1 &= \lambda\big[f(U_1(t-1)) + g(U_1(t-\theta))\big]U_1,\\
\dot{U}_2 &= \lambda\big[f(U_2(t-1)) + g(U_2(t-\theta))\big]U_2 + D(U_1 + U_3 - 2U_2),\\
\dot{U}_3 &= \lambda\big[f(U_3(t-1)) + g(U_3(t-\theta))\big]U_3 + D(U_1 + U_2 - 2U_3).
\end{aligned} \qquad (2)$$

Обычно системы (1) или (2) рассматриваются в предположении, что λ велико. Однако в настоящей работе мы изучим ситуацию с помощью локальных методов. Возможность аналитического асимптотического исследования системы (2) имеется при условии близости ее параметров к критическим в задаче об устойчивости состояния равновесия системы (2).

Предполагая дополнительно, что $f'(0) = -1$, $g'(0) = 0$, и полагая, что в цепи связи имеется дополнительное запаздывание (см. также [10,11]), приходим к задаче, близкой к системе, состоящей из трех уравнений Хатчинсона [12] с запаздывающей вещательной связью:

$$\begin{aligned}
\dot{N}_1 &= \lambda\big[1 - N_1(t-1)\big]N_1,\\
\dot{N}_2 &= \lambda\big[1 - N_2(t-1)\big]N_2 + D(N_1(t-h) + N_3(t-h) - 2N_2),\\
\dot{N}_3 &= \lambda\big[1 - N_3(t-1)\big]N_3 + D(N_1(t-h) + N_2(t-h) - 2N_3).
\end{aligned} \qquad (3)$$

В этой задаче $N_j(t)$, $j = 1, 2, 3$ можно интерпретировать как плотности численности трех близких популяций, $\lambda > 0$ — мальтузианский коэффициент линейного роста. Величина $D > 0$ отвечает за интенсивность взаимодействия между популяциями. Параметр $h > 0$ представляет собой запаздывание в цепи связи. Всюду далее предполагается, что связь между популяциями слабая, т.е. параметр D мал. Отметим, что влияние запаздывания в цепи связи между осцилляторами рассматривалось также в статьях [10, 11].

Система (3) моделирует ситуацию, когда популяции слабо связаны между собой, например, разделены географически. При этом одна из популяций может влиять на обе оставшиеся, которые в свою очередь способны влиять друг на друга, но не влияют на первую.

Уравнение Хатчинсона, впервые предложенное в [12], является простейшим способом учета возрастной структуры в задаче о динамике популяции особей, борющихся за общую пищу. Изучению уравнения Хатчинсона посвящено большое количество публикаций, в частности, задача диффузионного взаимодействия осцилляторов типа уравнения Хатчинсона исследовалась в [4, 13, 14]. В настоящей работе изучается динамика системы из трех уравнений Хатчинсона со слабой вещательной связью между ними. Локальная динамика трех осцилляторов общего вида, связанных вещательным образом без запаздывания, рассмотрена в [15]. В настоящей работе исследуется влияние запаздывания, введенного в цепь связи, на динамику изучаемой системы.

Сдвинемся в системе (3) в единичное состояние равновесия $N_j = 1+u_j$, параметр λ выберем равным $\pi/2+\varepsilon$, где ε – малый положительный параметр, а параметр связи D предполагается пропорциональным малому параметру $D = \varepsilon d$, $d > 0$ – величина порядка единицы. В результате получим систему

$$\begin{aligned}
\dot{u}_1 &= -\left(\tfrac{\pi}{2} + \varepsilon\right) u_1(t-1)(1+u_1), \\
\dot{u}_2 &= -\left(\tfrac{\pi}{2} + \varepsilon\right) u_2(t-1)(1+u_2) + \varepsilon d(u_1(t-h) + u_3(t-h) - 2u_2), \\
\dot{u}_3 &= -\left(\tfrac{\pi}{2} + \varepsilon\right) u_3(t-1)(1+u_3) + \varepsilon d(u_1(t-h) + u_2(t-h) - 2u_3),
\end{aligned} \tag{4}$$

в которой при $\varepsilon = 0$ в спектре устойчивости имеется пара чисто мнимых собственных чисел $\mu = \pm i\pi/2$ кратности 3, которой соответствуют три линейно независимые собственные функции.

При помощи локального асимптотического анализа и иллюстрирующего его численного эксперимента изучим периодические, квазипериодические и хаотические решения системы (4).

2. Построение нормальной формы

Учитывая, что в рассматриваемой системе (4) выбрана вещательная структура связи между осцилляторами, первый элемент системы односторонним образом воздействует на два других, которые, в свою очередь, взаимодействуют друг с другом. В силу этого, рассмотрим первое уравнение системы (4) отдельно от двух оставшихся. Устойчивое периодическое решение первого уравнения имеет следующую асимптотику:

$$u_1(t)=\sqrt{\varepsilon}\sqrt{\frac{40}{3\pi-2}}\cos\Big(\Big(\frac{\pi}{2}-\varepsilon\frac{2}{3\pi-2}\Big)t+c\Big)+$$
$$+\varepsilon\frac{4\sqrt{5}}{3\pi-2}\cos\Big(\Big(\pi-\varepsilon\frac{4}{3\pi-2}\Big)t-\gamma+2c\Big)+O(\varepsilon^{3/2}),\quad(5)$$

где $\gamma=\arctan(1/2)$, а c — произвольная константа (см., например, [16]).

Для дальнейшего локального анализа воздействия друг на друга осцилляторов u_2 и u_3 применим стандартный метод нормальных форм [16]. Воспользуемся заменой:

$$u_j(t)=\sqrt{\varepsilon}(z_j(\tau)e^{i\frac{\pi}{2}t}+\overline{z}_j(\tau)e^{-i\frac{\pi}{2}t})+\varepsilon u_{j1}(t,\tau)+\varepsilon^{3/2}u_{j2}(t,\tau)+\dots,\quad(6)$$

где $z_j(\tau)$, $(j=2,3)$ — комплекснозначные функции медленного времени $\tau=\varepsilon t$, которые определяются из условий разрешимости соответствующих задач в классе периодических функций.

Учитывая разложение

$$z_j(\tau-\varepsilon)=z_j(\tau)-\varepsilon z_j'(\tau)+\dots,$$

а также асимптотику (5), на третьем шаге алгоритма из условий разрешимости задач для $u_{j2}(t,\tau)$ в классе 4–периодических по t функций получаем следующую нормальную форму:

$$\begin{aligned}\Big(1+i\frac{\pi}{2}\Big)z_2'&=iz_2+\frac{(1-3i)\pi}{10}z_2|z_2|^2+d\Big(z_1e^{-i\frac{\pi}{2}h}+z_3e^{-i\frac{\pi}{2}h}-2z_2\Big),\\ \Big(1+i\frac{\pi}{2}\Big)z_3'&=iz_3+\frac{(1-3i)\pi}{10}z_3|z_3|^2+d\Big(z_1e^{-i\frac{\pi}{2}h}+z_2e^{-i\frac{\pi}{2}h}-2z_3\Big).\end{aligned}\quad(7)$$

Перейдем в системе (7) к полярным переменным. Для этого выполним замены $z_j=\sqrt{\frac{10}{3\pi-2}}\xi_je^{i\varphi_j}$ и $\tau=\frac{2\pi}{\pi^2+4}s$, $j=1,2,3$ и введем ряд обозначений: $\alpha=\varphi_2-\varphi_1$, $\beta=\varphi_3-\varphi_1$ и $h^*=\pi h/2$. В итоге получим нормальную форму, записанную в амплитудных и фазовых переменных:

$$\begin{aligned}\xi_2'&=\xi_2-\xi_2^3+d^*\big(\cos(\alpha-\delta^*)+\xi_3\cos(\beta-\alpha+\delta^*)-2\xi_2\cos(\delta^*+h^*)\big),\\ \xi_3'&=\xi_3-\xi_3^3+d^*\big(\cos(\beta-\delta^*)+\xi_2\cos(\beta-\alpha-\delta^*)-2\xi_3\cos(\delta^*+h^*)\big),\\ \alpha'&=b(1-\xi_2^2)-d^*\big(\frac{1}{\xi_2}\sin(\alpha-\delta^*)-\frac{\xi_3}{\xi_2}\sin(\beta-\alpha+\delta^*)+2\sin(\delta^*+h^*)\big),\\ \beta'&=b(1-\xi_3^2)-d^*\big(\frac{1}{\xi_3}\sin(\beta-\delta^*)+\frac{\xi_2}{\xi_3}\sin(\beta-\alpha-\delta^*)+2\sin(\delta^*+h^*)\big),\end{aligned}\quad(8)$$

где $b=(\pi+6)/(3\pi-2)$, $d^*=d\sqrt{\pi^2+4}/\pi$, $\delta^*=-\arctan(\pi/2)-h^*$. (Подробное изложение процесса перехода к амплитудным и фазовым переменным приведено в работе [17]).

Отметим, что для построенной нормальной формы (7) имеет место теорема о соответствии ее грубых режимов решениям исходной системы (4) той же устойчивости с асимптотикой (5), (6) (см., например, [16]). Однако в случае, если система (7) обнаруживает сложное хаотическое поведение, такую теорему обосновать не удается. В связи с этим, кроме аналитических методов исследования уместно применение численного анализа сложных неупорядоченных колебательных режимов, реализующихся в рассматриваемой системе.

3. Некоторые частные случаи. Анализ редуцированной системы

Рассмотрим частный случай однонаправленного воздействия одного осциллятора u_1 на другой (u_2 или u_3). В полученной нормальной форме (8) инвариантному многообразию $u_2 = u_3$ исходной системы (4) соответствует плоскость, заданная равенствами $\xi_2 = \xi_3$, $\alpha = \beta$. На этой плоскости система (8) приводится к следующему виду:

$$\begin{aligned} \xi' &= \xi(1 + d^* \cos\delta^* - 2d^* \cos(\delta^* + h^*) - \xi^2) + d^* \cos(\alpha - \delta^*), \\ \alpha' &= b(1 - \xi^2) - d^* \Big(\frac{1}{\xi} \sin(\alpha - \delta^*) - \sin\delta^* + 2\sin(\delta^* + h^*)\Big), \end{aligned} \tag{9}$$

где $\xi = \xi_2 = \xi_3$.

В работе [17] найдены состояния равновесия редуцированной системы (9) и условия их устойчивости. Кроме того, описана последовательность бифуркаций, происходящих на фазовом цилиндре системы (9) при уменьшении параметра d^*. В случае $h = 0$ показано, что при значениях параметра $d^* > d_0 \approx 1.1599$ глобально устойчивым является единственное, с точностью до добавки к переменной α периода, состояние равновесия $(1, 0)^T$. При $d^* = d_0$ рождается пара состояний равновесия, одно из которых устойчиво, а второе — неустойчиво. В дальнейшем при уменьшении значения параметра d^* до критического $d_1 \approx 1.0033$ неустойчивая неподвижная точка сливается с состоянием равновесия $(1, 0)^T$ и отбирает его устойчивость. При значении $d_{\text{кр.}} \approx 0.8536$ с устойчивым состоянием равновесия происходит бифуркация Андронова–Хопфа и оно теряет устойчивость с рождением устойчивого цикла. При дальнейшем уменьшении параметра d^* родившийся устойчивый цикл увеличивается в размерах и при значении $d_S \approx 0.6717$ сливается с петлей сепаратрисы седловой точки $(1, 0)^T$ и исчезает. При значениях $0 < d^* < 0.6717$ система имеет одно устойчивое состояние равновесия и пару неустойчивых неподвижных точек.

Также в статье [17] показано, что при относительно малых положительных значениях параметра h^* сценарий фазовых перестроек качественно не меняется. Отличия от изложенного выше сценария заключаются лишь в следующем: родившийся в результате бифуркации Андронова–Хопфа устойчивый цикл сливается с петлей сепаратрисы седловой точки $(1, 0)^T$ и при этом рождается цикл, охватывающий фазовый цилиндр. Это означает, что у системы (9) имеется устойчивое решение с периодической амплитудной переменной $\xi(t)$ и бегущей фазовой переменной $\alpha(t)$. При дальнейшем уменьшении параметра d^* цикл, охватывающий фазовый цилиндр, исчезает, и остается устойчивым лишь одно состояние равновесия.

Учитывая утверждение о соответствии между грубыми режимами нормальной формы и исходной задачи, найденная у нормальной формы (9) особенность, состоящая в наличии цикла с бегущей фазовой переменной, соответствует тому, что у системы (4) при $h \neq 0$ могут существовать двухчастотные колебания с набегом рассогласования фазы между ведущим и ведомыми осцилляторами.

4. Численный анализ неупорядоченных колебательных режимов

Система (8) при отсутствии вещающего осциллятора ($u_1(t) \equiv 0$) сводится к трехмерной и подробно изучена в работах [4, 13]. При изменении бифуркационного параметра d^* в ней происходят фазовые перестройки, не связанные с хаотическим поведением. Наличие вещающего осциллятора и запаздывания в цепи связи приводит к тому, что в системе реализуются сложные неупорядоченные колебательные режимы. Как было отмечено ранее, если система (7) обнаруживает сложное хаотическое поведение, то стандартное утверждение о соответствии ее грубых режимов решениям исходной системы (4) обосновать не удается. В связи с этим уместно применение численных методов исследования динамики рассматриваемых систем.

Перейдем к описанию полученных результатов.

Хорошее представление о фазовых перестройках, происходящих в системе (8), может дать зависимость старшего ляпуновского показателя λ_{max} от значения бифуркационного параметра d^* (см. [18]). Выберем значение $h^* = 0.5$ и приведем графики зависимости показателя λ_{max} от d^* на различных интервалах его изменения.

На рис. 1 видно, что промежутки с хаотическим поведением решений несколько раз сменяют периодические.

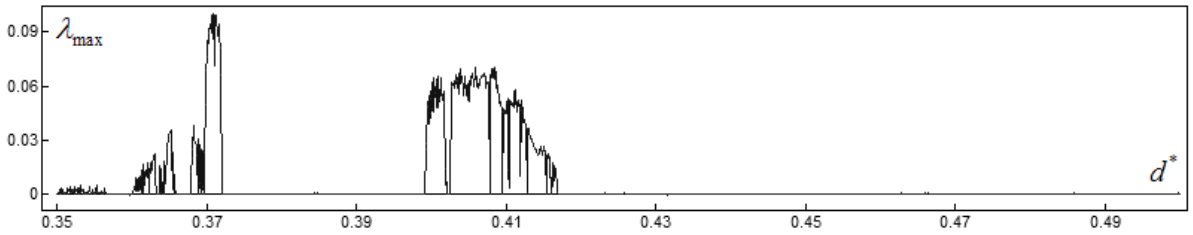


Рис. 1. Зависимость старшего ляпуновского показателя λ_{max} от параметра d^* на отрезке $[0.35; 0.5]$

Fig. 1. The dependence of the maximal Lyapunov exponent λ_{max} from the parameter d^* at interval $[0.35; 0.5]$

На рис. 2 и рис. 3 приведены увеличенные фрагменты отрезка $[0.35; 0.5]$, представляющие наибольший интерес для численного исследования.

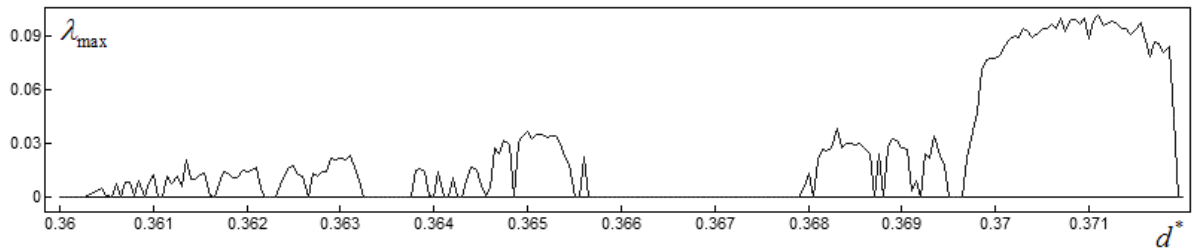


Рис. 2. Зависимость старшего ляпуновского показателя λ_{max} от параметра d^* на отрезке $[0.36; 0.372]$

Fig. 2. The dependence of the maximal Lyapunov exponent λ_{max} from the parameter d^* at interval $[0.36; 0.372]$

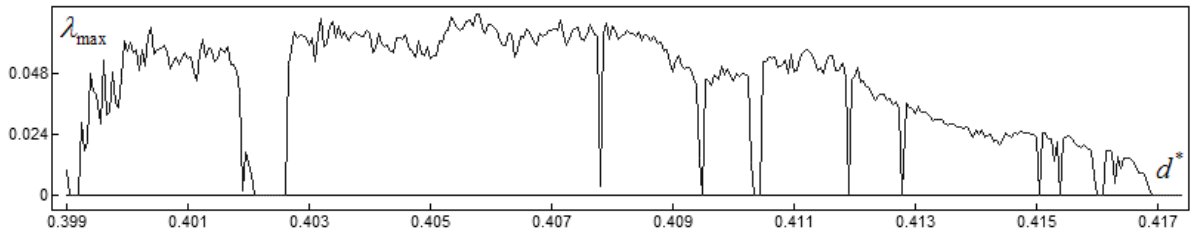


Рис. 3. Зависимость старшего ляпуновского показателя λ_{max} от параметра d^* на отрезке $[0.399; 0.42]$
Fig. 3. The dependence of the maximal Lyapunov exponent λ_{max} from the parameter d^* at interval $[0.399; 0.42]$

Перейдем теперь к более подробному описанию общего сценария фазовых перестроек. Зафиксируем значение запаздывания $h^* = 0.5$, тогда параметр δ^* оказывается равен $\delta^* \approx -1.50365$. Отметим, что величина оставшегося параметра системы (8) не зависит от h и равна $b \approx 1.23122$. На рис. 4–8 представлены проекции траекторий системы (8) на плоскость $\xi_1 O \xi_2$. При уменьшении бифуркационного параметра d^*, в системе (8) наблюдаются следующие фазовые перестройки:

1. При $d^* > d_1 \approx 0.622$ глобально устойчиво состояние равновесия.

2. При $d^* = d_1$ от состояния равновесия ответвляется самосимметричный устойчивый цикл (бифуркация Андронова–Хопфа) (см. рис. 4).

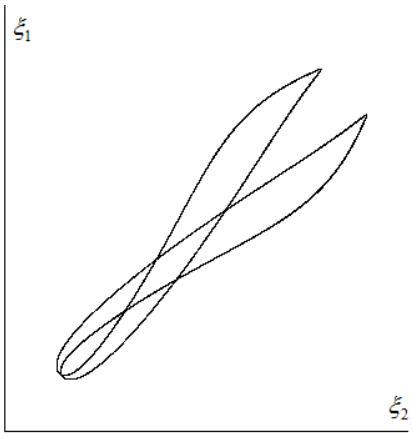


Рис. 4. Самосимметричный устойчивый цикл при $d^* = 0.465$
Fig. 4. Self-symmetric stable cycle at $d^* = 0.465$

3. При $d^* = d_2 \approx 0.435$ указанная симметрия цикла теряется, он расщепляется на два симметричных цикла (бифуркация потери симметрии) (см. рис. 5).

4. При $d^* = d_{31} \approx 0.418$; $d_{32} \approx 0.4172$; $\ldots$; $d_{3\infty} \approx 0.415$ с каждым из циклов происходят бифуркации удвоения периода (см. рис. 6). В результате при $d^* < d_{3\infty}$ имеем два симметричных странных аттрактора, возникших по фейгенбаумовскому сценарию (см. рис. 7).

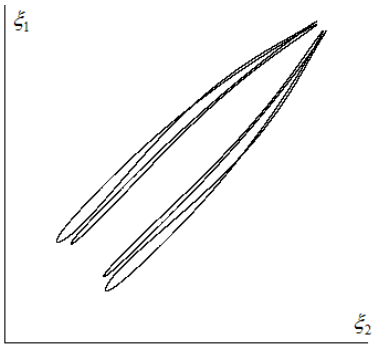
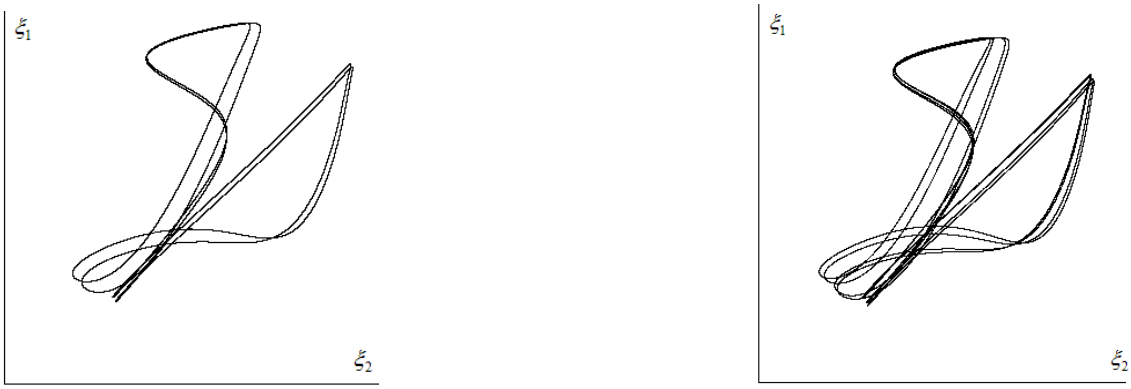


Рис. 5. Пара симметричных циклов при $d^* = 0.43$
Fig. 5. A pair of symmetric cycles at $d^* = 0.43$



a) b)

Рис. 6. Бифуркации удвоения периода для одного из симметричных циклов:
a) при $d^* = 0.418$; b) при $d^* = 0.417$
Fig. 6. Period doubling bifurcations for one of the symmetric cycles:
a) at $d^* = 0.418$; b) at $d^* = 0.417$

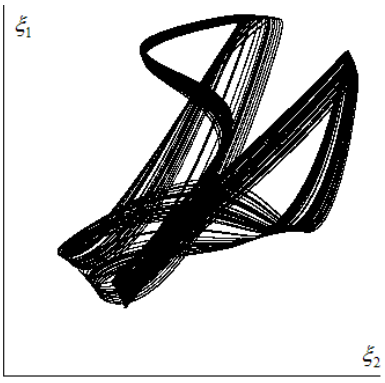
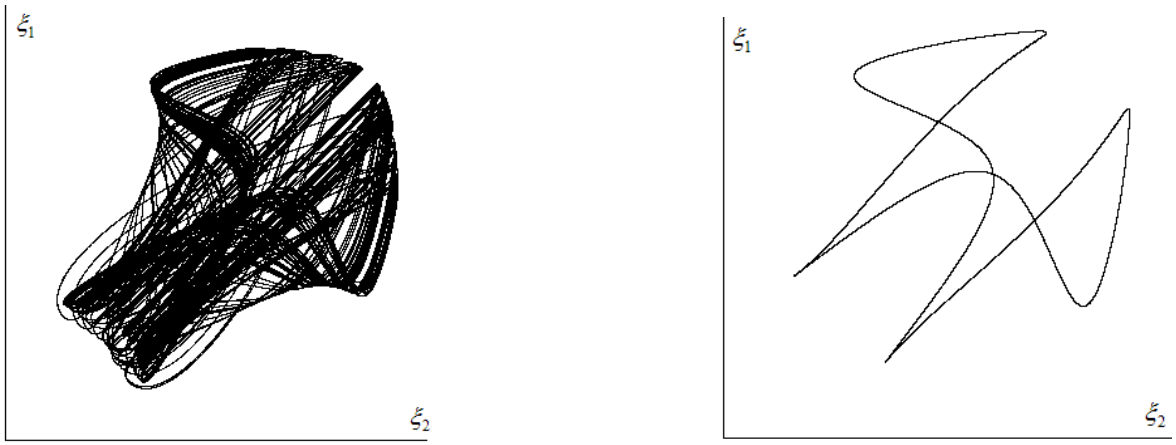


Рис. 7. Вид одного из пары симметричных странных аттракторов при $d^* = 0.415$
Fig. 7. One of a pair of symmetric strange attractors at $d^* = 0.415$

5. При $d^* = d_4 \approx 0.413$ пара симметричных странных аттракторов объединяется в один самосимметричный, который при $d^* = d_5 \approx 0.398$ превращается в самосимметричный двухобходный цикл (см. рис. 8).



a) b)

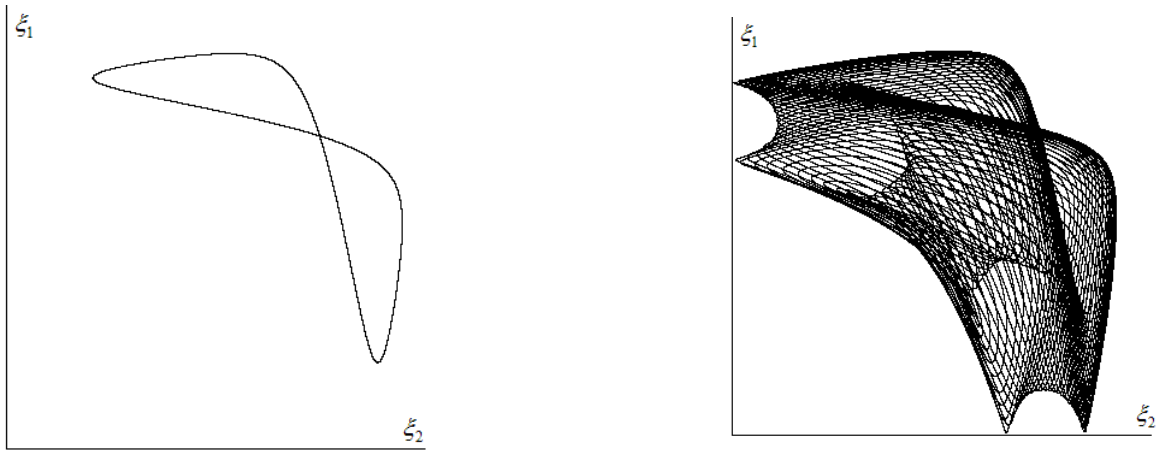
Рис. 8. a) Самосимметричный хаотический аттрактор при $d^* = 0.413$;
b) Самосимметричный двухобходный цикл при $d^* = 0.398$
Fig. 8. a) Self-symmetric chaotic attractor at $d^* = 0.413$;
b) Self-symmetric double-pass cycle at $d^* = 0.398$

6. При уменьшении d^* процесс повторяется: теряется симметрия цикла, затем с каждым из пары родившихся циклов происходят бифуркации удвоения, завершающиеся рождением симметричных странных аттракторов и т. д.

Таким образом, получается каскад бифуркаций странных аттракторов и циклов, содержащий в себе в том числе и каскад бифуркаций удвоения периода. Отметим, что близкий сценарий фазовых перестроек ранее уже встречался и описан в работе [4] для нормальной формы (8) без вещательного осциллятора и при относительно большом b.

Отметим, что при $d^* < 0.356$ бифуркации в системе (8) связаны с фазовыми перестройками циклов и торов. При $0 < d^* < 0.328$ в фазовом пространстве системы имеется самосимметричный устойчивый цикл. При $d^* \approx 0.328$ он теряет устойчивость колебательным образом с образованием двумерного инвариантного тора. Эта бифуркация проиллюстрирована на рисунке 9.

Выполненный в работе численный эксперимент показал, что сценарии фазовых перестроек, связанные с бифуркациями сложных неупорядоченных колебаний, реализуются у системы (8) и при других значениях запаздывания h^*. Вместе с тем, область хаотического поведения, не пропадая вовсе, существенно сужается при изменении параметра h^* в ту или другую сторону.



a) b)

Рис. 9. a) Самосимметричный устойчивый цикл при $d^* = 0.3$; b) Тор при $d^* = 0.35$
Fig. 9. a) Self-symmetric stable cycle at $d^* = 0.3$; b) Torus at $d^* = 0.35$

5. Заключение

Рассмотренная в работе модель взаимодействия трех осцилляторов с вещательной запаздывающей связью между ними встречается в целом ряде нейродинамических и биофизических приложений. Динамические свойства нормальной формы, построенной в окрестности состояния равновесия системы при значениях параметров, близких к критическим, исследовались на основе сочетания аналитических и численных методов. Вычисление ляпуновских экспонент аттракторов нормальной формы позволило судить о сценарии фазовых перестроек, происходящих в системе, при изменении параметра связи. Наиболее примечательным является полученный на этом пути сценарий фазовых перестроек, содержащий на каждом этапе каскады бифуркаций удвоения, бифуркации расщепления сепаратрис и потери симметрии. Следует отметить, что для грубых режимов нормальной формы выполнено утверждение о соответствии, согласно которому устойчивому циклу (тору) нормальной формы соответствует двумерный тор (тор на единицу более высокой размерности) у исходной системы. В случае хаотических колебаний это утверждение, вообще говоря, не выполнено. Остается открытым интересный вопрос о сохранении у исходной системы самого каскада фазовых перестроек, наблюдаемого у нормальной формы.

Список литературы / References

[1] Alon U., “Network motifs: theory and experimental approaches”, *Nature Reviews Genetics*, **8**:6 (2007), 450–461.

[2] Milo R., Shen-Orr S., Itzkovitz S., et al., “Network motifs: Simple Building Blocks of Complex Networks”, *Science*, **298**:5594 (2002), 824–827.

[3] Yechiam Yemini, *The Topology of Biological Networks*, Computer Science Department, Columbia University, 2004.

[4] Глызин С. Д., “Динамические свойства простейших конечноразностных аппроксимаций краевой задачи «реакция-диффузия»”, *Дифференциальные уравнения*, **33**:6

(1997), 805–811; English transl.: Glyzin S. D., "Dynamical properties of the simplest finite-difference approximations of the "reaction-diffusion" boundary value problem", *Differential Equations*, **33**:6 (1997), 808–814.

[5] Глызин С. Д., Колесов А. Ю., Розов Н. Х., "О явлениях хаоса в кольце из трех однонаправленно связанных генераторов", *Журнал вычислительной математики и математической физики*, **46**:10 (2006), 1809 – 1821; English transl.: Glyzin S. D., Kolesov A. Yu., Rozov N. Kh., "Chaos Phenomena in a Circle of Three Unidirectionally Connected Oscillators", *Computational Mathematics and Mathematical Physics*, **46**:10 (2006), 1724 – 1736.

[6] Глызин С. Д., "Поведение решений нормальной формы системы трех связанных разностных автогенераторов", *Модел. и анализ информ. систем*, **13**:1 (2006), 49 – 57; [Glyzin S. D., "Solutions behavior of the normal form of a system of three coupled difference oscillators", *Model. Anal. Inform. Syst.*, **13**:1 (2006), 49 – 57, (in Russian)].

[7] Глызин С. Д., "Разностные аппроксимации уравнения «реакция-диффузия» на отрезке", *Модел. и анализ информ. систем*, **16**:3 (2009), 96 – 116; [Glyzin S. D., "Difference approximations of "reaction–diffusion" equation on a segment", *Model. Anal. Inform. Syst.*, **16**:3 (2009), 96 – 116, (in Russian)].

[8] Глызин С. Д., "Релаксационные колебания электрически связанных нейроподобных осцилляторов с запаздыванием", *Модел. и анализ информ. систем*, **17**:2 (2010), 28 – 47; [Glyzin S. D., "Relaxation oscillations of electrically coupled neuron-like systems with delay", *Model. Anal. Inform. Syst.*, **17**:2 (2010), 28 – 47, (in Russian)].

[9] Глызин С. Д., Колесов А. Ю., Розов Н. Х., "Релаксационные автоколебания в сетях импульсных нейронов", *Успехи математических наук*, **70**:3(423) (2015), 3–76; English transl.: Glyzin S. D., Kolesov A. Yu., Rozov N. Kh., "Self-excited relaxation oscillations in networks of impulse neurons", *Russian Math. Surveys*, **70**:3 (2015), 383–452.

[10] Глызин С. Д., Киселева Е. О., "Учет запаздывания в цепочке связи между осцилляторами", *Модел. и анализ информ. систем*, **17**:2 (2010), 133–143; [Glyzin S. D., Kiseleva E. O., "The account of delay in a connecting element between two oscillators", *Model. Anal. Inform. Syst.*, **17**:2 (2010), 133–143 (in Russian)].

[11] Глызин С. Д., Солдатова Е. А., "Фактор запаздывания и десинхронизация колебаний связанных осцилляторов ФитцХью–Нагумо", *Модел. и анализ информ. систем*, **17**:3 (2010), 134–143; [Glyzin S. D., Soldatova E. A., "The factor of delay in a system of coupled oscillators FitzHugh–Nagumo", *Model. Anal. Inform. Syst.*, **17**:3 (2010), 134–143, (in Russian)].

[12] Hutchinson G. E., "Circular causal system in ecology", *Ann. N.-Y. Acad. Sci.*, **50** (1948), 221–246.

[13] Глызин С. Д., "Стационарные режимы одной конечноразностной аппроксимации уравнения Хатчинсона с диффузией", *Качественные и приближенные методы исследования операторных уравнений*, 1986, 112–127; [Glyzin S. D., "Statsionarnyye rezhimy odnoy konechnoraznostnoy approksimatsii uravneniya Khatchinsona s diffuziyey", *Kachestvennyye i priblizhennyye metody issledovaniya operatornykh uravneniy*, 1986, 112–127, (in Russian)].

[14] Глызин С. Д., "Учет возрастных групп в уравнении Хатчинсона", *Модел. и анализ информ. систем*, **14**:3 (2007), 29–42; [Glyzin S. D., "A registration of age groups for the Hutchinson's equation", *Model. Anal. Inform. Syst.*, **14**:3 (2007), 29–42, (in Russian)].

[15] Толбей А. О., "Локальная динамика трех осцилляторов со связью вещательного типа", *Модел. и анализ информ. систем*, **19**:3 (2012), 105–112; [Tolbey A. O., "Local Dynamics of Three Coupled Oscillators with a Feedback Loop", *Model. Anal. Inform. Syst.*, **19**:3 (2012), 105–112, (in Russian)].

[16] Глызин С. Д., Колесов А. Ю., *Локальные методы анализа динамических систем*, ЯрГУ, Ярославль, 2006, 92 с.; [Glyzin S. D., Kolesov A. Yu., *Lokalnye metody analiza dinamicheskih sistem*, Yaroslavl State University, Yaroslavl, 2006, 92 pp., (in Russian)].

[17] Марушкина Е. А., "Периодические и квазипериодические решения в системе трех уравнений Хатчинсона с запаздывающей вещательной связью", *Модел. и анализ информ. систем*, **25**:1 (2018), 102–111; [Marushkina E. A., "Periodic and Quasiperiodic

Solutions in the System of Three Hutchinson Equations with a Delayed Broadcast Connection", *Model. Anal. Inform. Syst.*, **25**:1 (2018), 102–111, (in Russian)].

[18] Глызин Д. С., Глызин С. Д., Колесов А. Ю., Розов Н. Х., "Метод динамической перенормировки для нахождения максимального ляпуновского показателя хаотического аттрактора", *Дифференциальные уравнения*, **41**:2 (2005), 268–273; English transl.: Glyzin D. S., Glyzin S. D., Kolesov A. Yu., Rozov N. Kh., "The dynamic renormalization method for finding the maximum lyapunov exponent of a chaotic attractor", *Differential Equations*, **41**:2 (2005), 284–289.

Glyzin S. D., Marushkina E. A., "Disordered Oscillations in a Neural Network of Three Oscillators with a Delayed Broadcast Connection", *Modeling and Analysis of Information Systems*, **25**:5 (2018), 572–583.

DOI: 10.18255/1818-1015-2018-5-572-583

Abstract. A model of neural association of three pulsed neurons with a delayed broadcast connection is considered. It is assumed that the parameters of the problem are chosen near the critical point of stability loss by the homogeneous equilibrium state of the system. Because of the broadcast connection the equation corresponding to one of the oscillators can be detached in the system. The two remaining impulse neurons interact with each other and, in addition, there is a periodic external action, determined by the broadcast neuron. Under these conditions, the normal form of this system is constructed for the values of parameters close to the critical ones on a stable invariant integral manifold. This normal form is reduced to a four-dimensional system with two variables responsible for the oscillation amplitudes, and the other two, defined as the difference between the phase variables of these oscillators with the phase variable of the broadcast oscillator. The obtained normal form has an invariant manifold on which the amplitude and phase variables of the oscillators coincide. The dynamics of the problem on this manifold is described. An important result was obtained on the basis of numerical analysis of the normal form. It turned out that periodic and chaotic oscillatory solutions can occur when the coupling between the oscillators is weakened. Moreover, a cascade of bifurcations associated with the same type of phase rearrangements was discovered, where a self-symmetric stable cycle alternately loses symmetry with the appearance of two symmetrical cycles. A cascade of bifurcations of doubling occurs with each of these cycles with the appearance of symmetric chaotic regimes. With further reduction of the coupling parameter, these symmetric chaotic regimes are combined into a self-symmetric one, which is then rebuilt into a self-symmetric cycle of a more complex form compared to the cycle obtained at the previous step. Then the whole process is repeated. Lyapunov exponents were calculated to study chaotic attractors of the system.

Keywords: oscillator, neural network, broadcasting connection, delay, stability, bifurcation, chaotic attractor

On the authors:
Sergey D. Glyzin, orcid.org/0000-0002-6403-4061, Doctor, Professor, P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia,
e-mail: glyzin@uniyar.ac.ru

Elena A. Marushkina, orcid.org/0000-0001-9183-6484, PhD, Researcher, P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia,
e-mail: marushkina-ea@yandex.ru

Acknowledgments:
The reported study was funded by RFBR, according to the research project No. 16-31-60039 mol_a_dk.