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 22 № 6(60) 2015

Основан в 1999 году
Выходит 6 раз в год

Адрес редакции: ЯрГУ, ул. Советская, 14, г. Ярославль, 150000, 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.

СОДЕРЖАНИЕ

Моделирование и анализ информационных систем. Т. 22, №6. 2015

Свидетельство о регистрации СМИ ПИ №ФС77-49724 от 11.05.2012 выдано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций. Учредитель – Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования "Ярославский государственный университет им. П. Г. Демидова". Подписной индекс – 31907 в Каталоге российской прессы "Почта России". Редактор, корректор А.А. Аладьева. Редактор перевода Э.И. Соколова. Подписано в печать 28.12.2015. Дата выхода в свет 24.01. 2016. Формат 60x84$^1/_8$. Усл. печ. л. 15,34. Уч.-изд. л. 14,0. Объем 131 с. Тираж 50 экз. Свободная цена. Заказ Адрес типографии: ул. Советская, 14, оф. 109, г. Ярославль, 150000 Россия. Адрес издателя: Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, г. Ярославль, 150000 Россия.

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

P.G. Demidov Yaroslavl State University

MODELING AND ANALYSIS OF INFORMATION SYSTEMS

Volume 22 No 6(60) 2015

Founded in 1999
6 issues per year

Editorial Office Address: P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia
Website: http://mais-journal.ru, e-mail: mais@uniyar.ac.ru

Contents

Modeling and Analysis of Information Systems. Vol. 22, No 6. 2015

Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 735–749
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 735–749

DOI: 10.18255/1818-1015-2015-6-735-749

UDC 519.987

End-to-end Information Flow Security Model for Software-Defined Networks

Chaly D. Ju.[1], Nikitin E. S., Antoshina E. Ju., Sokolov V. A.[2]

Received October 21, 2015

Software-defined networks (SDN) are a novel paradigm of networking which became an enabler technology for many modern applications such as network virtualization, policy-based access control and many others. Software can provide flexibility and fast-paced innovations in the networking; however, it has a complex nature. In this connection there is an increasing necessity of means for assuring its correctness and security. Abstract models for SDN can tackle these challenges. This paper addresses to confidentiality and some integrity properties of SDNs. These are critical properties for multi-tenant SDN environments, since the network management software must ensure that no confidential data of one tenant are leaked to other tenants in spite of using the same physical infrastructure. We define a notion of end-to-end security in context of software-defined networks and propose a semantic model where the reasoning is possible about confidentiality, and we can check that confidential information flows do not interfere with non-confidential ones. We show that the model can be extended in order to reason about networks with secure and insecure links which can arise, for example, in wireless environments.

The article is published in the authors' wording.

Keywords: SDN, security, formal models

For citation: Chaly D. Ju., Nikitin E. S., Antoshina E. Ju., Sokolov V. A., "End-to-end Information Flow Security Model for Software-Defined Networks", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 735–749.

On the authors:
Chaly Dmitriy Jurevich, orcid.org/0000-0003-0553-7387, PhD,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: dmitry.chaly@gmail.com

Nikitin Evgeniy Sergeevich, orcid.org/0000-0002-2341-9950, student,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: nik.zhenya@gmail.com

Antoshina Ekaterina Jurevna, orcid.org/0009-0203-2514-7755, graduate student,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: kantoshina@gmai.com

Sokolov Valeriy Anatolevich, orcid.org/0000-0003-1427-4937, doctor of science,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: valery-sokolov@yandex.ru

Acknowledgments:
[1]This work was supported by the RFBR under the project 14-01-31539 mol-a.
[2]This work was supported by the Ministry of Education and Science of the Russian Federation under contract ID RFMEFI57414X0036.

Introduction

The traditional approach to the networking assumes that a network is constructed using vendor-specific hardware which is tightly coupled with a proprietary software which implements distributed protocols. Protocols can provide various services including topology discovery, routing, access control, quality of service and other features. Network operators must install these devices and configure every protocol they intend to use. This tight integration of forwarding and control functionality within proprietary devices restricts innovations and slows down introduction of new network services to modern networks. Bringing open standards and programmability to networks are key points of introduction of software-defined networks (SDN).

Software-defined networks have drawn a lot of attention in recent years and provide a rich set of concepts for centralized management of modern networks. The main aim of SDNs is to provide general principles of packet forwarding and to decouple control software from forwarding devices. This makes it possible to bring innovations to networks without changing the underlying hardware just relying on a well-defined standard collection of packet-processing functions that forms the body of the OpenFlow standard [9]. Software controller provides a centralized management and orchestration of the whole network inspecting network packets and installing forwarding rules to switches under management.

However, the standard does not solve security problems which are the great challenge in todays networking [12]. The centralized control of SDN can benefit in enforcing security strategies, however, the lack of models makes this problem challenging [4]. We can discuss the security of SDN in three dimensions: integrity, availability and confidentiality. The integrity assumes that no data is corrupt due to internal or external events or misconfiguration. This problem was in the focus of study in [1] where the authors propose a model checking-based approach to find configuration inconsistencies that can lead to network partitioning. The availability property means that data are available when needed. At some extent this property is achieved by load balancing in SDN [8].

The confidentiality considers that secret data cannot be inferred by an attacker or unintentionally. This policy can be imposed by using access control lists, encryption etc. One of the recent attempts that introduce access control lists to SDN is [5]. However, access control does not prevent leaks of confidential data through improperly configured or buggy software [11]. The confidentiality property can be seen in a broad sense, so we focuse on the end-to-end confidentiality. We assume that an attacker can observe non-confidential entities of the network and has a limited access to the network infrastructure. The confidentiality can be achieved at some extent when network resources are separated from each other in slices [7], however, slices isolate flows in the network, thus, are too restrictive. Software nature of network control in SDN is a cause for a try to apply security methods that are developed for programming languages [11].

There is an extensive work on semantic foundations of networking programming languages that can provide a solid basis for reasoning about networks. One of the first attempts was Frenetic language [6] that provided abstractions for SDN programming and means for combining these abstractions in a meaningful and consistent way. The NetKAT project [2] defined a semantics that can help to prove reachability in networks (which is an integrity property) and address several security properties at once, however, the decision procedure for this formalism has PSPACE complexity. Focusing only on confidentiality may reduce complexity of verification. The confidentiality property was investigated for programming languages [11] and implemented for model [3] and industry-level languages [13]. This approach is based on rigorous semantic rules that impose restrictions on information flows in programming languages.

In this paper we propose a framework for checking confidentiality on-the-fly for modern SDNs that are conformed to the OpenFlow standard. We introduce a set of semantic rules that help us to verify that controller application does not allow non-confidential information flows. We assume that network consists of high and low security nodes and latter we extend our concept to a model of network that can contain secure and insecure links.

Consider a simple model of a software-defined network. Let us assume that the network consists of endpoints or hosts that generate data traffic and a set of unified intermediate nodes forwarding the traffic. These forwarding devices are OpenFlow switches that conform to standard [9]. There is a single node representing a controller application that manages all the switches by using secure channels. Thus, a network can be represented as a graph where the nodes are either hosts or switches, and the edges are links.

The OpenFlow switch contains a set of physical or logical *ports* which are interfaces for passing packets between the switch and the network. According to specification [10] the OpenFlow switch consists of an *OpenFlow channel*, one or more *flow tables*, and a *group table*. The OpenFlow channel is used for managing the switch and for passing relevant data about the traffic under management to the controller. Flow tables provide means for forwarding and processing packets. The controller can add, update or modify flow entries in flow tables. Such an entry consists of *match fields*, *counters* and a set of *instructions* to apply to matching packets. The group table enables additional methods of forwarding by representing a set of ports as a single entity. Thus, group tables do not represent a fundamentally different abstraction and can be modeled via flow tables. So, we exclude group tables out of consideration.

Each arriving packet is matched to flow table entries starting from the first one. If the match is found, instructions associated with this flow entry are executed. If the packet is mismatched to each table entry, the outcome depends on the table-miss flow entry. Such a packet can be passed to the controller, dropped or handed to the next flow table. We will assume that the packet is passed to the controller.

The match field is a predicate which partitions the set of all flows passing through the network. Standard [10] proposes that matching field is a conjunctive predicate where each conjunct can impose conditions on various packet headers including Ethernet, IP, TCP, etc. Each flow has a source and a destination host. Thus, the matching field can be modeled as $m_{src} \wedge m_{dst}$, where m_{src} and m_{dst} are conjuncts for matching the source and destination hosts of the flow, respectively.

Counters are variables that contain statistical information about flows. Since counters have no direct impact on forwarding, exclude them out of consideration.

Let us consider instructions that can be executed if a packet is matched to a flow table entry. The standard proposes that instructions are lists of actions. Some of these actions are required to be implemented by switch designers and the rest are optional.

The actions are executed in the order specified by the list and are applied immediately to the packet. We consider only the following actions:

- *Output(port).* This action specifies the port to which the associated packet will be forwarded.
- *Drop.* The packet can be discarded from the network using this action.
- *Set.* The optional set action allows to modify packet header fields, such as IP and MAC addresses, various tags, etc.
- *Delete.* This action deletes flow entries according to a match.

We limit ourselves to considering only listed actions when trying to capture most relevant OpenFlow processing features and not to overwhelm the model.

The packet processing model is the following. Upon receiving an incoming packet p, the controller emits an ordered list of match fields each of which is paired with an action. This list is installed to the switch.

The controller software implements specific network applications. There are a lot of them. For example, the controller can implement a simple hub application where it installs such forwarding rules to a switch, so an incoming packet is flooded to all switch ports except for an ingress port. Other applications include a learning switch, where the controller determines what subnets are reachable from different switch ports and it installs forwarding rules in such a manner that the incoming packet goes to a port from which its destination host is reachable, otherwise it is flooded. The controller can implement various security checking policies, for example, allowing to forward a packet from authenticated hosts and dropping packets from other hosts.

1. End-to-end Security Model for SDN

The controller application gathers all the information about the network under management. So, we can assume that the security level of each endpoint is known. The security level can be revealed using some kind of a protocol or can be defined ad-hoc. For the sake of simplicity we assume that there are two security levels of endpoints: *high* and *low.* Since a host is identified by the IP address, we can think that the controller can map the space of IP addresses of the network under management into a set of security levels. Denote a security level of a host h as $h : low$ or $h : high$.

For further discussion we need means for reasoning about sets of hosts. The network itself or its subnets aggregates hosts with different security levels. Define security predicates *exists* and $forall$ that will give us a security type for a set of hosts $\{h_1, \dots, h_n\}$ as shown in Fig. 1.

If the set of hosts is homogeneous, i.e. all hosts have the same security level, the predicate $forall$ can be typed with the same security type as any host in the set. On the other hand, the *exists* predicate is *high* only if the set contains a *high* host. This predicate can not be typed as *low* and it will be seen later that we only need to check a possibility to reach a high security host.

One of the primary functions of the controller is routing that is essentially reasoning about reachability in networks. Model the network as a graph, so we are forced with

$$\frac{\{h_1 : low, \ldots, h_n : low\}}{\vdash forall(h_1, \ldots, h_n) : low} \quad (1) \qquad \frac{\{h_1 : high, \ldots, h_n : high\}}{\vdash forall(h_1, \ldots, h_n) : high} \quad (2)$$

$$\frac{\{h_1, \ldots, h_n \colon \exists h_i : high\}}{\vdash exists(h_1, \ldots, h_n) : high} \quad (3)$$

Fig 1. Security types for sets of hosts

deducing reachable hosts from a given switch. Thus, we define a function $reachable(s, p)$ that evaluates a set of hosts reachable from a switch s if we first go to the port p. It is not easy to calculate reachability in real networks since the network can be dynamic because of mobility of hosts and installed forwarding rules. However, a superset of the set of reachable hosts can be computed using breadth-first search on a network graph. More accurate algorithms that take into account network policies can be found in [2].

The data plane of the network is represented by switches that use flow tables for implementation of network policies. Each flow table entry contains a matching field that is modeled as a predicate $match = m_{src} \wedge m_{dst}$. We define the functions src and dst that map a predicate to a set of source and destination hosts, respectively, such that the predicate is true.

The next part of the model is a *packet processing context*. When the OpenFlow switch can not match the packet to any flow table entry, the model assumes that the packet is forwarded to the controller. The controller can examine headers of the packet and determine the host that emitted the packet. Security type of the host implies the packet processing context so we can analyze whether the controller generates a secure response to the packet or not.

A security-type system can help to reason about the security type of a single interaction between a switch and a controller. Figure 2 presents typing rules for instructions that can be installed to a switch s by the controller in response to a packet pkt. We can use the presented security-type system for inferring a type of the interaction. If the type can be inferred, the interaction is secure, otherwise it allows leaks of confidential data.

Let us consider a proposed set of typing rules in more detail. Rule 4 assigns a type for a packet processing context in such a way that the context pc agrees with the security type of the source host of the packet pkt. The packet processing context is a virtual action in the list formed by the controller.

For the *Drop* action (rule 5) we strictly isolate flows of different security levels, that is, the source host of the flow must correspond to the context of the action. Such a type setting prevents interference between packet processing contexts and actions of different security levels. Violation of this can lead to a covert channel when low hosts discover that a high host installs *Drop* action by observing occasional drops. Setting low type to the *Drop* action ensures that under the low security packet processing context a drop can occur only for low security flows. Non-interference property holds even if we allow a low security packet processing context to drop high security flows since no information about high security flows can be inferred. However, we discard this and guarantee that integrity for high security flows can not be broken by low hosts.

$$\frac{\vdash forall(src(pkt)) : pc}{[pc] \vdash pkt} \quad (4) \qquad \frac{\vdash forall(src(match)) : pc}{[pc] \vdash match \times Drop} \quad (5)$$

$$\frac{\vdash exists(dst(match)) : high \quad \vdash exists(reachable(s, port)) : high}{[high] \vdash match \times Output(port)} \quad (6)$$

$$\frac{\vdash forall(src(match)) : low}{[low] \vdash match \times Output(port)} \quad (7)$$

$$\frac{\vdash forall(dst(match)) : high}{[high] \vdash match \times Delete} \quad (8) \qquad \frac{\vdash forall(src(match)) : low}{[low] \vdash match \times Delete} \quad (9)$$

$$\frac{\vdash forall(src(match)) : low \quad \vdash forall(src(pattern)) : low}{[low] \vdash match \times Set(pattern)} \quad (10)$$

$$\frac{\begin{array}{l}\vdash forall(src(match)) : high \\ \vdash forall(dst(pattern)) : high \\ \vdash forall(src(pattern)) : high\end{array}}{[high] \vdash match \times Set(pattern)} \quad (11) \qquad \frac{[pc] \vdash A \, [pc] \vdash B}{[pc] \vdash A; B} \quad (12)$$

Fig 2. Security-type system for SDN

The *Output(port)* action type depends heavily on the matching condition (rules 6–7). If the match forwards traffic to high security hosts, there must be a high security host reachable from the *port*. In this case the security context of *Output(port)* is high. If the source of the traffic is a low security host, it can be forwarded anywhere and the security context of this action is low. The *Output(port)* action can not be typed if the match condition specifies that traffic from high security hosts must be forwarded to a low security host. If this is the case, $forall(src(match))$ can not be typed as *low* and $exists(dst(match))$ can not be typed as *high* implying that premises for both rules do not hold.

Rules 8–9 for *Delete* action guarantee that the eviction of flows from the flow table of the switch is done in the respective security context. So, a low packet can not be a reason to remove high matches and vice versa.

Rule 10 guarantees that any low security flow can not become a high security flow by changing the source address of the packet. By imposing this condition we achieve a certain level of integrity since a low packet can not become a high packet that may later influence other high security flows. Rule 11 assures that a high security flow stays high providing no information leak to the low security plane. In both rules we denote as a *pattern* the data that have to be written to the packet header.

The controller can respond with several actions at once, thus we must have means for inferring a security type for a list of actions. This can be done using rule 12 that assigns a security type for a composition. Here, A and B can be either single actions or lists of actions.

The proposed rules constitute a security-type system which describes what security type must be assigned to a list of actions. This list of actions is formed by a controller in response to a packet incoming from the switch. The packet specifies the first action in the list called a packet processing context. If the whole list can be typed using the proposed security-type system, the list ensures non-interference among flows of different security levels and fulfills some integrity properties.

This security-type system can be further extended to SDNs with insecure links. We can define an insecure link as a channel that can not be trusted since they are exposed to everyone like Wi-Fi medium or may be public channels shared by various tenants. This setting leads to a new confidentiality violation since high data traffic may be forwarded to an insecure link. It could be noted that any link can be secured using traffic encryption. We propose the following extension to our model. Let us assume that every link has a security level (*high* for secure links and *low* for insecure ones) and it is known to the controller. It is the same that we did for endpoints. Also we must provide means of reasoning about secure paths in the network.

Let us define a function $reachable_s(s, port)$ that calculates a set of hosts that are reachable via paths such that every link in the path is secure. Since the controller has the information about the network graph, it can be done using breadth-first search or taking into account current network policies [2].

Since a confidentiality flaw can occur when high traffic is forwarded to an insecure link, we must only refine rule 6 that is used for inferring the type for *Output* action considering high traffic. We propose the following change:

$$\frac{\vdash exists(dst(match)) : high \quad \vdash exists(reachable_s(s, port)) : high}{[high] \vdash match \times Output(port)} \tag{13}$$

Thus, we allow high traffic only to those switch ports that start with a secure link and have the possibility to reach the destination host using a secure path.

This shows that the proposed model can be used as a basis for reasoning about various aspects of confidentiality in software-defined networks.

2. An Example of the Model Application

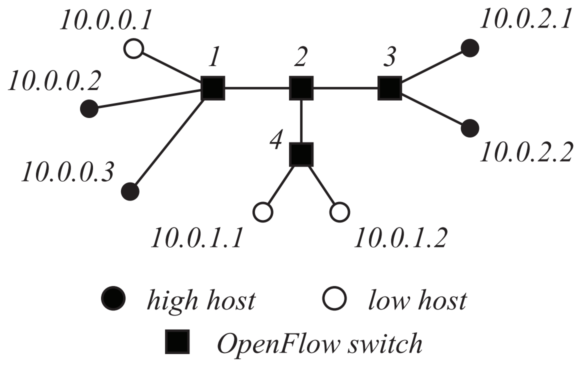


Fig 3. A sample network with high and low security hosts

We consider a learning switch application as an example. The switches in the network initially have no flow entries and forward incoming packets to the controller. The controller examines each packet and stores in the internal database the source address of the packet along with the port from where it was received. The port and packet headers are forwarded to the controller as an OpenFlow *packet in* message. Next time the switch receives the packet destined to the address that was seen earlier, the controller can infer the port to which the packet must be forwarded. If the port can not be determined, the packet is flooded to all the switch ports.

Algorithm 1 Learning switch algorithm

```
1:  $pkt \leftarrow$ packet arrived to the controller
2:  $port \leftarrow$ from which port $pkt$ received
3:  if find(src($pkt$)) is null then
4:      push (src($pkt$), $port$)
5:  end if
6:  $fport \leftarrow$ find(dst($pkt$))
7:  if $fport$ is null then
8:      for all switch port $i$ other than $port$ do
9:          emit (src($pkt$),dst($pkt$))$\times$Output($i$)
10:     end for
11: else
12:     emit (src($pkt$), dst($pkt$))$\times$Output($fport$)
13:     emit (dst($pkt$), src($pkt$))$\times$Output($port$)
14: end if
```

A simple algorithm for the learning switch is shown as Algorithm 1. The input data for the algorithm is an incoming packet *pkt* and the port *port* from which it has been received. The controller maintains an internal database which can be implemented as a hash which supports the following operations:

- *push(address, port)*. The operation creates a mapping between the *address* and the *port* in the internal database.

- *find(address)*. This is a query to the database which returns port number associated with *address* and *null* if there is no such an association.

There is an **emit** operator in our language which appends the action to the list of instructions destined to the switch. The list is sent to the switch when the algorithm is stopped. Then we can analyze the list and find if it is secure or not.

Algorithm 1 checks whether a mapping between a source address of *pkt* and *port* exists. If there is no such mapping, it writes it in lines 3–5. Thereafter, we try to find if we have learned the port to which we can forward the packet *pkt* (line 6). If no such a port exists then we flood the packet to all ports except ingress port (lines 8–10). Otherwise, we emit forwarding rules which set up a duplex channel between source and destination hosts of the packet (lines 12–13). We assume that entries responsible for flooding packets will be eventually evicted from switches and replaced by direct forwarding entries.

Recall the network from Fig. 3. Assume that the controller database is empty and there is no forwarding rules at switches, so each switch sends a *packet in* message to the controller upon a packet receipt. The security flaw arises even when the first packet travels from any high security host. For example, if the host 10.0.0.2 sends a packet *pkt* to the host 10.0.2.1, the following list of rules will be emitted by the controller to the switch 1 according to lines 8–10 of Algorithm 1:

$$\begin{gathered}(10.0.0.2, 10.0.2.1) \times Output(1)\\(10.0.0.2, 10.0.2.1) \times Output(3)\\(10.0.0.2, 10.0.2.1) \times Output(4)\end{gathered}$$

The first instruction installs the rule which forwards all packets from high security host 10.0.0.2 to a low security host 10.0.0.1. Let us try to discover a security type of packet *pkt* processing.

First, by rule 2 we can infer that

$$\frac{10.0.0.2 : high}{\vdash forall(\{10.0.0.2\}) : high}$$

Since $src(pkt) = \{10.0.0.2\}$ using rule 4, the following holds

$$\frac{\vdash forall(\{10.0.0.2\}) : high}{[high] \vdash pkt}$$

Next, we should discover the type of the action $(10.0.0.2, 10.0.2.1) \times Output(1)$. Let us denote as $match = (10.0.0.2, 10.0.2.1)$, $src(match) = \{10.0.0.2\}$, $dst(match) = \{10.0.2.1\}$ and the $reachable(s, port) = \{10.0.0.1\}$. Thus,

$$\vdash exists(src(match)) : high$$

but

$$\nvdash exists(reachable(s, port)) : high$$

$$\nvdash forall(\{10.0.0.2\}) : low$$

so the premises for rule 6 not hold.

Likewise,

$$\nvdash \forall(src(match)) : low,$$

hence we can not infer the only premise for rule 7. Thus, the considered action can not be typed, so the whole list can not be typed.

Algorithm 2 proposes an enhanced version of the learning switch. This version is free from many security leaks but let us analyze it formally. The algorithms breaks into two parts. The first one is represented by lines 8–19 where packets from low sources are processed. If the output port can not be identified, the packet is flooded to all ports of the switch (lines 9–11), otherwise forwarding rules are installed to the switch. These rules include the one which redirects packet *pkt* to the destination host (line 13 and another which either create a channel with the opposite direction (line 15) or sets the action to *Drop* if the opposite forwarding rule forms a route from high host to low host (line 17). The second part of the algorithm processes packets from high sources (lines 21–31). If the destination for such a high packet is a low host, we drop the packet (line 22). If the

Algorithm 2 Secure learning switch algorithm

```
1: $pkt \leftarrow$ packet arrived to the controller
2: $port \leftarrow$ from which port $pkt$ received
3: if find(src($pkt$)) is null then
4:    push(src($pkt$), $port$)
5: end if
6: $fport \leftarrow$find(dst($pkt$))
7: if src($pkt$):$low$ then
8:    if fport is null then
9:       for all switch port $i$ other than $port$ do
10:          emit (src($pkt$), dst($pkt$))×Output($i$)
11:       end for
12:    else
13:       (src($pkt$), dst($pkt$))×Output($fport$)
14:       if (dst($pkt$):$low$) then
15:          emit (dst($pkt$), src($pkt$))×Output($port$)
16:       else
17:          emit (dst($pkt$), src($pkt$))×Drop
18:       end if
19:    end if
20: else
21:    if dst($pkt$):$low$ then
22:       emit (src($pkt$), dst($pkt$))×Drop
23:    else
24:       if $fport$ is null then
25:          for all switch port $i$ other than $port$ and $exists(i)$ : $high$ do
26:             emit (src($pkt$), dst($pkt$))×Output($i$)
27:          end for
28:       else
29:          emit (src($pkt$), dst($pkt$))×Output($fport$)
30:          emit (dst($pkt$), src($pkt$))×Output($port$)
31:       end if
32:    end if
33: end if
```

controller does not find the port to forward the packet, the packet is flooded but only to high ports (lines 25–27), otherwise forwarding rules are installed (lines 29–30).

Let us show how security properties of Algorithm 2 can be proved. If the condition in line 8 is true, the following holds for lines 8–19 by rule 1:

$$\frac{\{src(pkt) : low\}}{\vdash forall(src(pkt)) : low}$$

And by rule 4:

$$\frac{\vdash forall(src(pkt)) : low}{[low] \vdash pkt}.$$

Assume that $fport$ is null, the packet must be flooded to all ports except $port$ (lines 9–11). So, the controller emits *packet out* messages which can be typed using rule 7:

$$\frac{\vdash forall(src(pkt)) : low}{[low] \vdash (src(pkt), dst(pkt)) \times Output(i)}.$$

Applying rule 12, we have

$$\frac{[low] \vdash pkt \quad [low] \vdash (src(pkt), dst(pkt)) \times Output(i)}{[low] \vdash pkt; (src(pkt), dst(pkt)) \times Output(i)}.$$

Thus, the whole list of emitted actions is typed and these actions are safe.

Assume that $fport$ is not null, then the controller emits an action in line 13 which safety can be ensured using the same inference as in flooding case above. The second action of the list depends on the security type of dst(pkt). If it is low, the action in line 15 is emitted. The security type of the action is the following:

$$match = (dst(pkt), src(pkt))$$

$$\frac{\{src(match) : low\}}{\vdash forall(src(match)) : low} \quad \text{(rule 1)}$$

$$\frac{forall(src(match)) : low}{\vdash [low] \vdash (src(pkt), dst(pkt)) \times Output(port)} \quad \text{(rule 7)}.$$

Thus, the security type of all emitted actions agree, so the whole list can be typed as *low*. If dst(pkt) is high (line 17), only the following can be inferred:

$$match = (dst(pkt), src(pkt))$$

$$\frac{\{src(match) : high\}}{\vdash forall(src(match)) : high} \quad \text{(rule 2)}$$

$$\frac{\vdash forall(src(match)) : high}{[high] \vdash match \times Drop} \quad \text{(rule 5)}.$$

This means that the security type of the *Drop* action from line 17 does not agree with the security type of previous actions and the packet processing context which are *low*. Thus, the *Drop* action can not be considered safe. Indeed, low packets must not trigger packet drops originated from high security hosts. If we carefully examine the code, we will see that such a drop is made in line 22 when the packet processing context is high. Hence, we can remove line 17 from our algorithm without harming the learning switch functionality.

If the packet *pkt* is originated from a high security host, Algorithm 2 proceeds to lines 21–31. The packet processing context is now *high*:

$$\frac{\{src(pkt) : high\}}{\vdash forall(src(pkt)) : high} \quad \text{(rule 2)}$$

$$\frac{\vdash forall(src(pkt)) : high}{[high] \vdash pkt} \quad \text{(rule 4).}$$

In this case three possibilities can occur:

1. A *Drop* action is emitted (line 22):

$$match = (src(pkt), dst(pkt))$$

$$\frac{\vdash forall(src(match)) : high}{[high] \vdash match \times Drop} \quad \text{(rule 5).}$$

2. The packet is flooded by using the list of *Output* actions (lines 25–27):

$$\vdash exists(i) : high \quad \text{(condition in line 25)}$$

$$match = (src(pkt), dst(pkt)),$$

since condition in line 21 does not hold

$$\frac{\{dst(match) : high\}}{\vdash exists(dst(match)) : high},$$

So, using rule 6 we can obtain

$$\frac{\vdash exists(dst(match)) : high \quad \vdash exists(i) : high}{[high] \vdash match \times Output(i)}.$$

3. Bidirectional forwarding is set (lines 29–30). Since both src(pkt):$high$ and dst(pkt):$high$ are fulfilled and it was determined that such packets came from ports *port* and *fport*, respectively, we can conclude that $exists(port) : high$ and $exists(fport) : high$. Using the same as shown earlier we can obtain that both *Output* actions are typed as $[high]$.

Thus, in all three cases the emitted actions are typed as *high*. This agrees with the packet processing context, and we can conclude that the whole list of emitted actions must be typed as $[high]$. That is the list is safe.

We have considered all the cases and all lists of actions the controller can install to a switch. We found a case where a packet from a low security flow can trigger packet drops from a high security flow. This shows that the proposed approach can find very subtle security discrepancies. In the context of our application this can not be considered as a security flaw, but it can lead to security leaks in more general settings.

3. Conclusion

Security is challenging in networking and must be further investigated for software-defined networks. There is a lack of formal models for making security analysis [4] and the paper proposes the approach that is based on a formal security-type system. This system ensures that the controller application does not violate security properties such as confidentiality and, at some extent, integrity. We have extended the proposed system so that can verify new confidentiality properties in case of insecure network links. The security system can be implemented as a software module of the controller and check whether network applications violate security properties.

There are both theoretic and practical challenges when considering SDN security. It is interesting to explore soundness and completeness of the proposed type system. Another fascinating problem is to introduce other security-type systems that have been recently developed for programming languages using the proposed approach for achieving a solid theoretical basis for static security analysis that can prove properties of an SDN controller at the compilation stage.

References

[1] E. Al-Shaer, S. Al-Haj, "FlowChecker: configuration analysis and verification of federated OpenFlow infrastructures", SafeConfig 2010 : 2nd ACM Workshop on Assurable and Usable Security Configuration (October 4, 2010, Chicago, IL, USA), 37–44.

[2] C. J. Anderson et al., "NetKAT: semantic foundations for networks", POPL 2014: 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (January 22–24, 2014, San Diego, USA), 113–126.

[3] E. Ju. Antoshina et al., "A translator with a security static analysis feature of an information flow for a simple programming language.", *Autom. Control and Comp. Sciences*, **48**:7 (2014), 589–593.

[4] M. Casado, N. Foster, A. Guha, "Abstractions for software-defined networks", *Communications of the ACM*, **57**:10 (2014), 86–95.

[5] M. Casado et al., "Ethane: taking control of the enterprise", ACM SIGCOMM 2007: Data Communications Festival (Augest 27–31, 2007, Kyoto, Japan).

[6] N. Foster et al., "Frenetic: a network programming language", The 16th ACM SIGPLAN International Conference on Functional Programming (September 19–21, 2011, Tokyo, Japan), 279–291.

[7] S. Gutz et al., "Splendid isolation: a slice abstraction for software-defined networks", ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN) (August 13, 2012, Helsinki, Finland), 2012, 79–84.

[8] C.-Y. Hong et al., "Achieving high utilization with software-driven WAN", ACM SIGCOMM 2013 (August 12 – 16, 2013, Hong Kong, China).

[9] N. McKeown et al., "OpenFlow: enabling innovation in campus networks", *ACM Comp. Comm. Review*, **38**:2 (2008), 69 – 74.

[10] Open Networking Foundation, "OpenFlow switch specification v. 1.4.0", *URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf*, Last accessed: 10.05.2015.

[11] A. Sabelfeld, A.C. Myers, "Language-based information-flow security", *IEEE Journal on Selected Areas in Communications*, **21** (2003), 5–19.

[12] R. Smeliansky, "SDN for network security", Modern Networking Technologies: SDN & NFV – The Next Generation of Computational Infrastructure (October 28–29, 2014, Moscow, Russia), 155–159.

[13] D. Zhang et al., "Jif: Java+ information flow", *URL: http://www.cs.cornell.edu/jif/*, Last accessed: 10.05.2015.

[14] D. Zhang et al., "A Hardware Design Language for Timing-Sensitive Information-Flow Security", ASPLOS 2015 : Architectural Support for Programming Languages and Operating Systems (Mar 14 – 18, 2015, Istanbul, Turkey).

[15] D. Hedin et al., "JSFlow: Tracking Information Flow in JavaScript and its APIs", The 29th Symposium On Applied Computing (March 24 – 28, 2014, Gyeongju, Korea), 1663–1671.

[16] O. Arden et al., "Sharing Mobile Code Securely With Information Flow Control", *IEEE Symp. on Security and Privacy (SP)*, 2012, 191–205.

[17] A. Cheung et al., "Using Program Analysis to Improve Database Applications", *IEEE Data Eng. Bull.*, **37**:1 (2014), 48–59.

DOI: 10.18255/1818-1015-2015-6-735-749

Модель безопасности информационных потоков для программно-конфигурируемых сетей

Чалый Д. Ю.[1], Никитин Е. С., Антошина Е. Ю., Соколов В. А.[2]

получена 21 октября 2015

Программно-конфигурируемые сети (ПКС, SDN, Software-defined Networks) являются новой парадигмой организации сетей, которая используется во многих современных приложениях, таких как виртуализация сети, управление доступом на основе политик безопасности и многих других. Программное обеспечение ПКС обеспечивает гибкость и быстрый темп инноваций в сети, однако оно имеет сложную природу, в связи с чем возникает необходимость в средствах обеспечения его корректности и безопасности. Абстрактные модели для ПКС могут решить эти задачи. Данная работа направлена на разработку моделей безопасного взаимодействия в ПКС, акцентируя внимание на таких свойствах безопасности, как конфиденциальность и, частично, целостность. Это критические свойства безопасности многопользовательских сетей, поскольку программное обеспечение, управляющее сетью, должно гарантировать, что конфиденциальные данные одного пользователя не будут переданы другим (нежелательным) пользователям. Мы определили понятие сквозной безопасности в контексте ПКС и предложили семантическую модель, позволяющую сделать обоснованный вывод о соблюдении конфиденциальности, и мы можем проверить, что конфиденциальные информационные потоки не смешиваются с не конфиденциальными. Мы показываем, что модель может быть расширена до обоснования соблюдения конфиденциальности в сетях с безопасными и небезопасными каналами связи, которые могут возникнуть, например, в беспроводных средах.

Статья представляет собой расширенную версию доклада на VI Международном семинаре “Program Semantics, Specification and Verification: Theory and Applications”, Казань, 2015.

Статья публикуется в авторской редакции.

Ключевые слова: ПКС, безопасность, формальные модели

Для цитирования: Чалый Д. Ю., Никитин Е. С., Антошина Е. Ю., Соколов В. А., "Модель безопасности информационных потоков для программно-конфигурируемых сетей", *Моделирование и анализ информационных систем*, **22**:6 (2015), 735–749.

Об авторах:
Чалый Дмитрий Юрьевич, orcid.org/0000-0003-0553-7387, канд. физ.-мат. наук., доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия, e-mail: dmitry.chaly@gmail.com

Никитин Евгений Сергеевич, orcid.org/0000-0002-2341-9950 , студент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия, e-mail: nik.zhenya@gmail.com

Антошина Екатерина Юрьевна, orcid.org/0000-0003-1081-1758, аспирант,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия, e-mail: kantoshina@gmai.com

Соколов Валерий Анатольевич, orcid.org/0000-0003-1427-4937, доктор физ.-мат. наук, профессор,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия, e-mail: valery-sokolov@yandex.ru

Благодарности:
[1]Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта 14-01-31539 мол-а.
[2]Исследование выполнено при финансовой поддержке Минобрнауки в рамках контракта ID RFMEFI57414X0036.

Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 750–762
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 750–762

DOI: 10.18255/1818-1015-2015-6-750-762

UDC 004.4'2

Model Oriented Approach for Industrial Software Development

Drobintsev P. D., Kotlyarov V. P., Voinov N. V., Nikiforov I. V.

Received October 21, 2015

The article considers the specifics of a model oriented approach to software development based on the usage of Model Driven Architecture (MDA), Model Driven Software Development (MDSD) and Model Driven Development (MDD) technologies. Benefits of this approach usage in the software development industry are described. The main emphasis is put on the system design, automated code generation for large systems, verification, proof of system properties and reduction of bug density. Drawbacks of the approach are also considered. The approach proposed in the article is specific for industrial software systems development. These systems are characterized by different levels of abstraction, which is used on modeling and code development phases. The approach allows to detail the model to the level of the system code, at the same time store the verified model semantics and provide the checking of the whole detailed model. Steps of translating abstract data structures (including transactions, signals and their parameters) into data structures used in detailed system implementation are presented. Also the grammar of a language for specifying rules of abstract model data structures transformation into real system detailed data structures is described. The results of applying the proposed method in the industrial technology are shown.

The article is published in the authors' wording.

Keywords: model oriented approach; multilevel software models; model specification by control flow and data flow; model verification; substitutions saving the correctness of proved properties

For citation: Drobintsev P. D., Kotlyarov V. P., Voinov N. V., Nikiforov I. V., "Model Oriented Approach for Industrial Software Development", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 750–762.

On the authors:

Drobintsev Pavel Dmitrievich, orcid.org/0000-0003-1116-7765, PhD,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya str., 29, St.Petersburg, 195251, Russia, e-mail: drob@ics2.ecd.spbstu.ru

Kotlyarov Vsevolod Pavlovich, orcid.org/0000-0003-3973-5218, PhD,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya str., 29, St.Petersburg, 195251, Russia, e-mail: vpk@spbstu.ru

Voinov Nikita Vladimirovich, orcid.org/0000-0002-0140-1178, PhD,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya str., 29, St.Petersburg, 195251, Russia, e-mail: voinov@ics2.ecd.spbstu.ru

Nikiforov Igor Valerievich, orcid.org/0000-0003-0198-1886, PhD,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya str., 29, St.Petersburg, 195251, Russia, e-mail: igor.nikiforovv@gmail.com

1. Model based technologies

One of the most perspective approaches to modern software product creation is usage of model oriented technologies both for software development and testing. Such technologies are called MDA (Model Driven Architecture) [1,2], MDD (Model Driven Development) [2] and MDSD (Model Driven Software Development) [3]. All of them are mainly aimed to design and generation of application target code based on a formal model.

The article is devoted to specifics of model oriented approaches usage in design and generation of large industrial software applications. These applications are characterized by multilevel representation related to detailing application functionality to the level where correct code is directly generated.

The idea of model oriented approach is creation of multilevel model of application during design process. A set of possible models transformations is presented in Fig. 1. This model is iteratively specified and detailed to the level when executable code can be generated. On the design stage formal model specification allows using verification together with other methods of static analysis with goal to guaranty correctness of the model on early stages of application development.

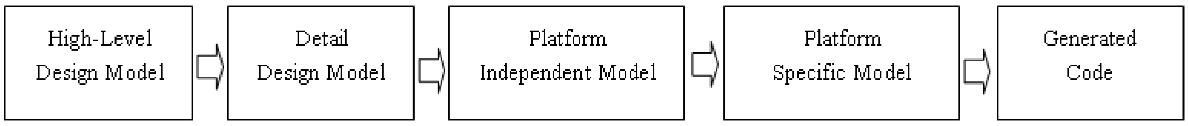


Fig. 1. Designing multilevel model of application

Statistics collected in companies which are using such approaches shows [4] that model-oriented techniques are usually used on system testing phase (up to 80% of projects) with the main goal - functional testing (up to 96%). The reason of such company's behavior is complexity of system testing for big industrial projects, which is based on huge efforts spent on quality guarantying [5]. To resolve this problem software developing companies are trying to reduce efforts for tests creation and simplify tests execution process. Usually reduction of testing efforts is linked to communication with customers because only customer of software has deep knowledge about domain specifics and model oriented approach helps to simplify such communications.

Researchers also consider that more than 80% [4] of model-oriented approaches use graphical notations, which simplifies working with formal notations for developers. Requirements for testers and customer representatives knowledge are reduced by this way and process of models developing is also simplified.

The following advantages of model-oriented approaches in comparison with manual test development methods can be found in research papers [4]:

- increasing productivity and reduction of efforts on development;
- systematic reuse of verified templates and solutions which leads to reduction of bugs density in generated code;
- analyzing and proving formal models properties on early stages of design.

Among drawbacks of the approach the following can be listed:

- different levels of detailing in multilevel formal model and real generated code which may lead to distortion of verified semantics during model detailing;
- complexity (impossibility in some cases) of aggregate proving the multilevel detailed model properties;
- complexity of multi criteria optimization while selecting balanced architecture of software application.

2. Drawbacks of models usage

The main drawback of formal models using in software products development is high level of model abstraction in comprising with a real system. At the stage of model design developers tend to specify only major behavior scenarios and data structures which affect the behavior, ignoring the implementation details. Usage of formal models on this stage allows to prove the correctness of system behavior in accordance with specifications, miscellaneous system's properties and to generate a set of test scenarios providing complete coverage for specified criteria.

As a model of the system and its implementation in the code vary significantly, the semantics of test scenarios generated from abstract model may differ from corresponding behavior observed in the real system. Therefore automatic check of system functioning correctness is impossible.

There are two ways to solve this issue.

The first one is developing of a detailed model which is as close to system program implementation as possible. In this case it is impossible to provide check of complete detailed model of industrial system (even of medium complexity) due to limitations of modern verification toolsets.

The second one is creating an abstract model of such complexity which does not prevent applying toolsets for proving behavioral properties. Further the abstract model can be detailed to the level of real system in such a way that proved properties will be spread on the detailed model. This method satisfies model based software development technology when applied iteratively and guarantees storing proved system properties up to code level.

When control flow is being detailed traditional elements of model control flow structuring can be used. Model fragments which shall be detailed are relocated into separate structural element (for example, an instance of class method). Then its analysis and formalization of its behaviors, which include specifying fragments of alternative and concurrent behavior, fragments of behavior limited by timer, fragments of behavior specific for exceptions and interruptions can be performed.

When data flow is being detailed formalization of new data structures, signals and transactions is performed. Each data structure can be represented by several nested structures of lower level. Signals in the system can be separated into several compound signals and actions of real system. New transactions can be added to the system to provide data consistency. Also detailing of one system signal into complete communication protocols between components becomes possible. This means that a communication protocol can be represented by only one signal on the abstract level while in the real code this protocol can be specified by a set of incoming and outgoing signals.

3. Levels of behavioral models development

One of high level languages for system formal model specification is Use Case Maps (UCM) [6, 8]. It provides visible and easy understandable graphical notation. Further abstract models will be specified in UCM language to demonstrate proposed approach in details. Also considered is VRS/TAT technology chain [7], which uses formal UCM models for behavioral scenarios generation.

Traditional steps of formal abstract model development in UCM language are the following:

1. Specifying main interacting agents (components) and their properties, attributes set by agent and global variables.
2. Introducing main system behaviors to the model and developing diagrams of agent's interaction control flow.
3. Developing internal behaviors for each agent and specifying data flow in the system.

Undoubted benefit of UCM language is possibility to create detailed structured behavioral diagrams. Structuring is specified both by Stub structural elements and reused diagrams (Maps), which are modeling function calls or macro substitution. Unfortunately, standard UCM language deals with primitive and abstract data structures, which are not enough to check implementation of a real system. This drawback is compensated by using metadata mechanism [8]. But metadata does not allow detailing data flow to more detailed levels. That's why for creating detailed behaviors it is proposed to use the following vertical levels of abstractions during behavioral models development (Fig. 2).

Another benefit of UCM usage is possibility to execute model verification process. UCM diagrams are used as input for VRS/TAT toolset which provides checks for specifications correctness. These checks can detect issues with unreachable states in the model, uninitialized variables in metadata, counterexamples for definite path in UCM, etc. After all checks are completed the user gets a verdict with a list of all findings and a set of counterexamples which show those paths in UCM model which lead to issue situations. If a finding is considered to be an error, the model is corrected and verification process is launched again. As a result after all fixes a correct formal model is obtained which can be used for further generation of test scenarios.

After formal model of a system has been specified in UCM language, behavioral scenarios generation is performed. Note that behavioral generator is based not on concrete values assigned to global variables and agents attributes, but on symbolic ones which reduces significantly the number of behavioral scenarios covering the model. However symbolic test scenarios cannot be used for applications testing as executing behavioral scenarios on the real system requires concrete values for variables. So the problem of different level of abstraction between model and real system still exists. In VRS/TAT technology concretization step [9] is used to convert symbolic test scenarios. On this step

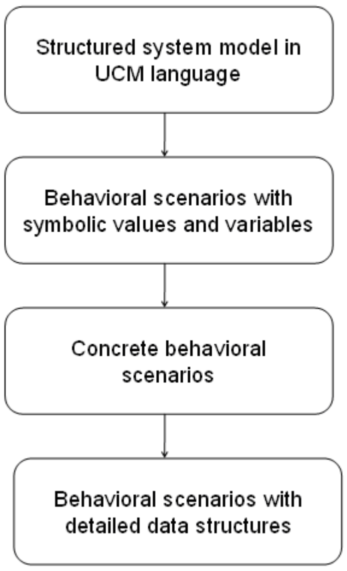
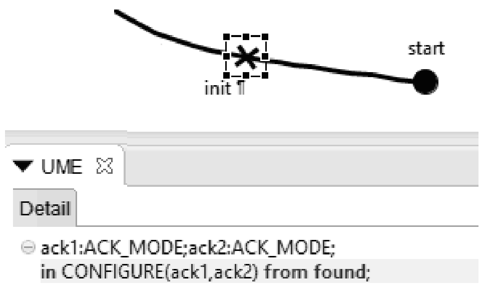

Structured system model in UCM language
Behavioral scenarios with symbolic values and variables
Concrete behavioral scenarios
Behavioral scenarios with detailed data structures

Fig. 2. Abstraction levels during developing behavioral scenarios

ranges of possible values for variables and attributes are calculated based on symbolic formula and symbolic values are substituted with concrete ones. But concretization of abstract model's behavioral scenarios is not enough for their execution, because on this stage scenarios still use abstract data structures which differ from data structures in real system. As a result conversion of concretized behavioral scenarios of abstract UCM level into scenarios of real system level was integrated into technology chain for behavioral scenarios generation.

4. Data structures conversion

In behavioral scenarios data structures are mainly used in signals parameters. Consider an example of converting signal structure of UCM level into detailed structures of real system for the signal "CONFIGURE".

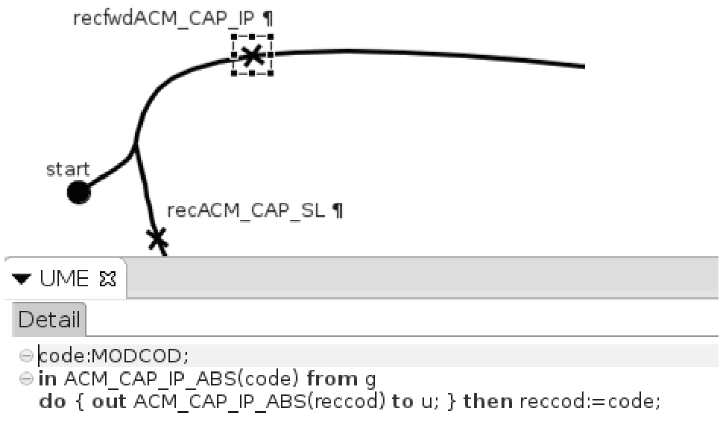


```
ack1:ACK_MODE;ack2:ACK_MODE;
in CONFIGURE(ack1,ack2) from found;
```

Fig. 3. Description of the "CONFIGURE" signal
in metadata of the "init" UCM element

In UCM model the element "init" contains metadata with the signal "CONFIGURE" and two signal parameters of UCM level: "ack1" and "ack2". Fig. 3 contains metadata of the UCM element "init" including description of a signal of UCM level.

There are two types of signals in UCM model: incoming to an agent and outgoing from an agent. Incoming signals are specified with the keyword "in" and can be sent either by an agent or from outside the system specifying with the keyword "found". Outgoing signals are specified with the keyword "out" and can be sent either to an agent or to outside the system specifying with the keyword "lost".



```
code:MODCOD;
in ACM_CAP_IP_ABS(code) from g
do { out ACM_CAP_IP_ABS(reccod) to u; } then reccod:=code;
```

Fig. 4. Description of the "ACM_CAP_IP_ABS" signal in metadata of the "recfwdACM_CAP_IP" UCM element

As an example in UCM model the element "recfwdACM_CAP_IP" contains metadata with the outgoing signal "ACM_CAP_IP_ABS" and the signal parameter "reccod" of UCM level. This signal shall be sent after the signal "ACM_CAP_IP_ABS" with the parameter "code" received from the agent "g". Fig. 4 contains metadata of the UCM element "recfwdACM_CAP_IP" including description of a signal of UCM level. The outgoing signal can be used only inside of "do" section as reaction of the system on some event.

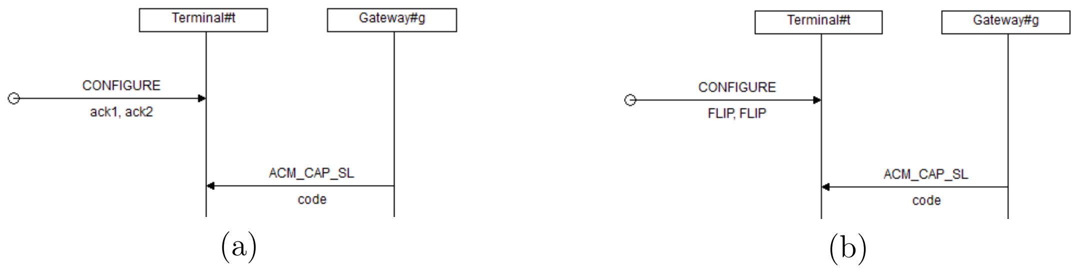
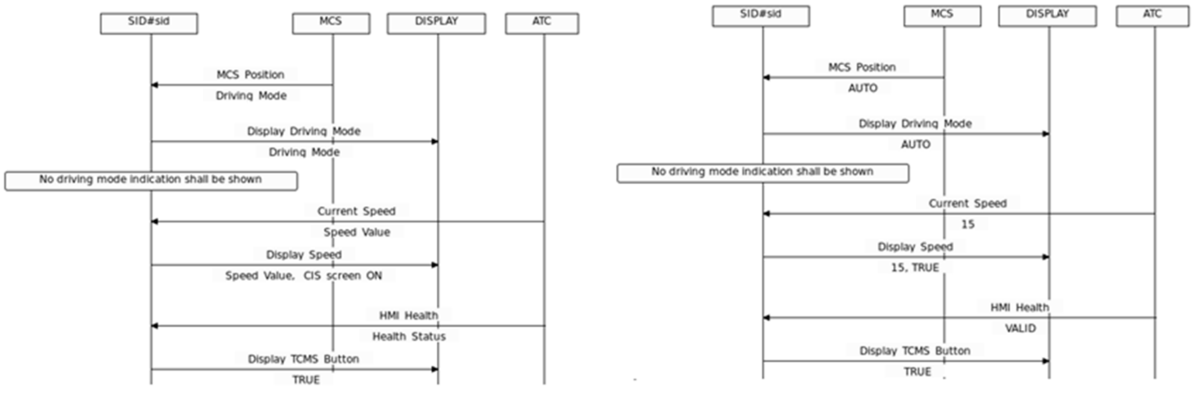


Fig. 5. Symbolic (a) and concrete (b) test scenarios containing the signal "CONFIGURED"

Based on high level UCM model symbolic behavioral scenarios are generated containing data structures described in metadata of UCM elements. Fig. 5(a) contains symbolic test scenario where the agent "Terminal#t" receives the signal "CONFIGURED". In symbolic scenarios actual names of UCM model agents specified in metadata are used. For example, the agent "Gateway#g" is the source of the signal "ACM_CAP_SL" and the agent "Terminal#t" is the destination. While the source of the signal "CONFIGURE" is outside the system.

Symbolic behavioral scenario is input data for concretization module which substitutes symbolic parameters with concrete values. In current example the parameters "ack1, ack2" are substituted with values "FLIP" and "FLIP". Fig. 5(b) contains concrete behavioral scenario. Fig. 6 contains another example of concretization where integer parameters are substituted together with parameters of string type. For example, the symbolic parameter "Speed Value" in the signals "Current Speed" and "Display Speed" (Fig. 6(a)) is concretized with value "15" (Fig. 6(b)).



(a) (b)

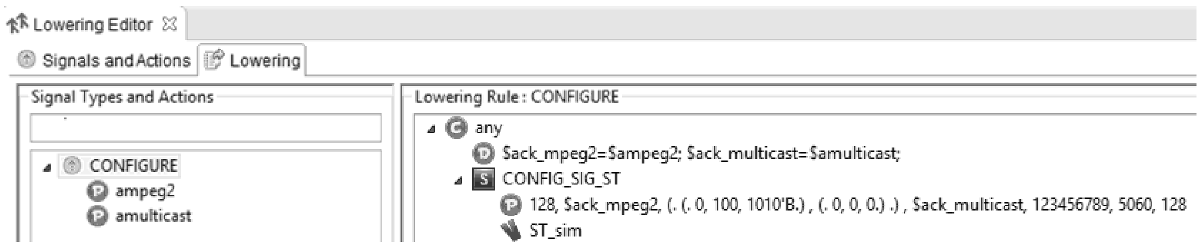
Fig. 6. Symbolic (a) and concrete (b) test scenarios
containing string and integer parameters

Note that after concretization interacting agents are not changed in any way. To convert concrete data structures into detailed structures a developer shall specify the rules of structures conversion: for each signal of UCM model a corresponding conversion condition (Lowering Condition) and detailed signal (Lowered Signal) are specified.

To keep proved system properties there are following limitations on the conversion:

- rules which allow separating constants into several independent parts (sets of variables) are prohibited;
- separating fields of variables values is prohibited;
- converting abstract signal into a protocol if this protocol is not represented by verified template is prohibited;
- only constant template values or values obtained at concretization step are allowed;
- violating consistent communication protocol is prohibited.

Fig. 7 contains the rule for converting the signal "CONFIGURED" into the signal "CONFIGURED_SIG_ST" for all occurrences of this signal in test scenario. This condition is specified by the keyword "any" in the rule.



Lowering Editor
Signals and Actions | Lowering

Signal Types and Actions
- CONFIGURE
  - ampeg2
  - amulticast

Lowering Rule : CONFIGURE
- any
  - $ack_mpeg2=$ampeg2; $ack_multicast=$amulticast;
  - CONFIG_SIG_ST
    - 128, $ack_mpeg2, (. (. 0, 100, 1010'B.) , (. 0, 0, 0.) .) , $ack_multicast, 123456789, 5060, 128
    - ST_sim

Fig. 7. Rule for converting the signal "CONFIGURED"
into the signal "CONFIG_SIF_ST"

Specification of conversion rules is based on the grammar of conversion language. Common view of the grammar for converting signals of abstract level into detailed level in BNF form is shown in Fig. 8.

```
LoweringSpec ::= UCMSignal "->" LoweringRule | LoweringSpec UCMSignal "->" LoweringRule
LoweringRule ::=  LoweringCondition | LoweringRule LoweringCondition
LoweringCondition ::= <condition STRING> ConditionContent
ConditionContent ::= LoweredElement | LoweredElement ConditionContent
LowredElement ::= LoweredDo | LoweredSignal | LoweredAction
LoweredDo ::= <code STRING>
LoweringSignal ::= <signal name STRING> SignalContent
SignalContent ::= ValueNotation Instance Via
ValueNotation ::= <empty> | <value STRING> | "(." ValueNotation ".)" | ValueNotation "," ValueNotation
Instance ::= <empty> | "TAT" | "SUT"
Via := <empty> | <port STRING>
UCMSignal ::= Name UCMParam
Name ::= <name STRING>
UCMParam ::= <empty> | <param name STRING> | UCMParam "," UCMParam
```

Fig. 8. Grammar of the conversion rules language

Based on the specified conversion rule each abstract signal in concrete behavioral scenario is processed and in case the signal satisfies to a conversion rule it is converted into detailed signal. Fig. 9 contains executable scenario with the detailed signal "CONFIG_-SIG_ST" which can be used for testing. Note that on this stage system agents are joined into two instances – TAT and SUT, which is required for testing process.

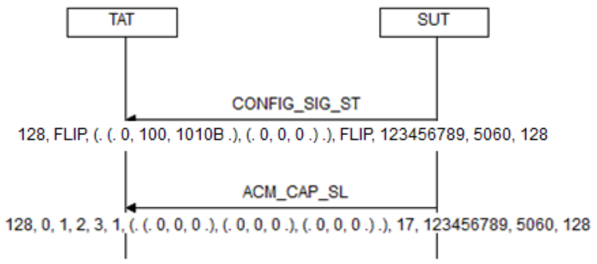


Fig. 9. The detailed signal "CONFIG_SIG_ST" of the real system

To exclude limitations on conversion of signals with templates usage the following techniques can be used. In case of some particular signal is converted into a set of signals (protocol) which are used as parameters variables verified on previous phase then combined conditions shall be used. In Fig. 10 (a) a signal with four parameters is presented. Consider that based on a template the signal shall be substituted into a set of signals.

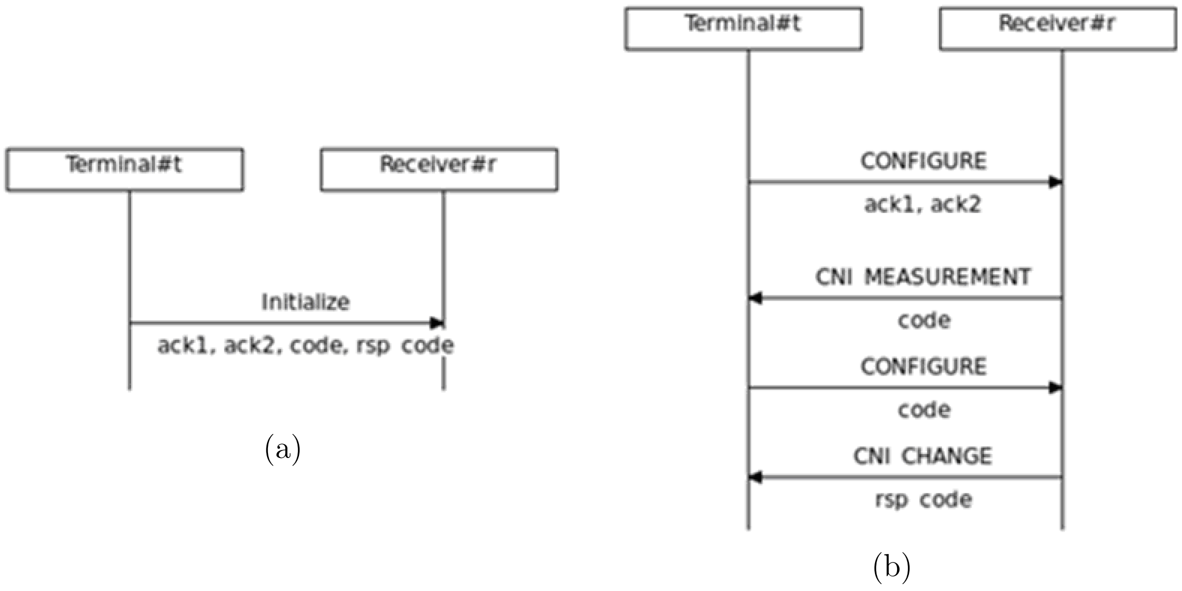


Fig. 10. Initial signal before (a) and after (b) conversion

Fig. 10 (b) contains a diagram with substituted signal. To verify correctness of such substitution a special filters shall be added into target code of application and test.

Another case when signal parameter is separated into parameters of two signals as presented in Fig. 11 (a,b).

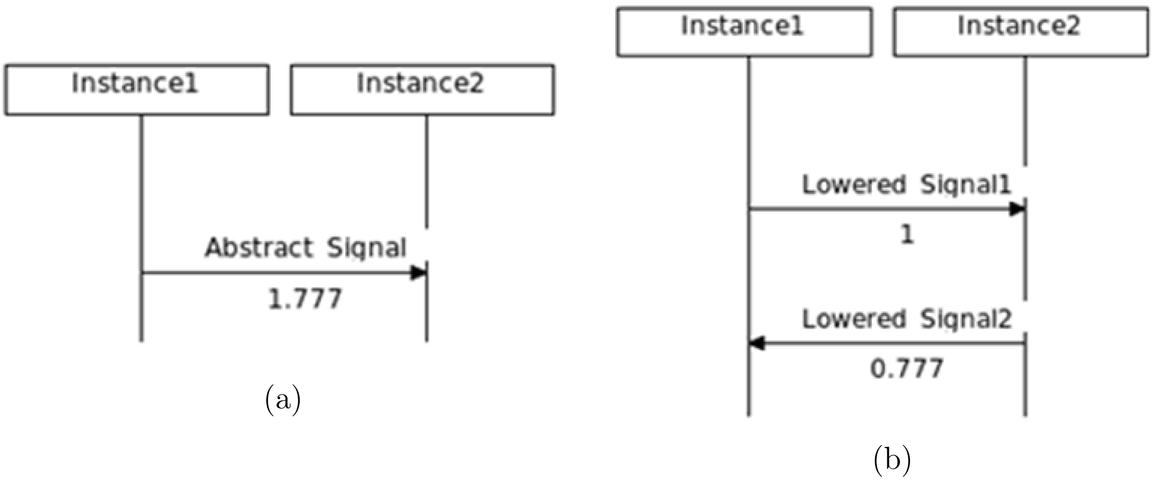


Fig. 11. Initial signal before (a) and after (b) parameters separation

The same solution with filter in target code of application and test can be used. The filter shall check that correctness of the model was not broken via generation of ranges for parameters of separated signals.

Usage of approach with filters allows to raise limitations of lowering conversion connected to maintenance of model correctness.

5. Overall scheme of conversion

Implemented module of behavioral scenarios conversion takes the concrete behavioral scenarios and specified rules of conversion as an input and the output is behavioral scenarios of the real system level which can be used for testing. Overall scheme of conversion is shown in Fig. 12.

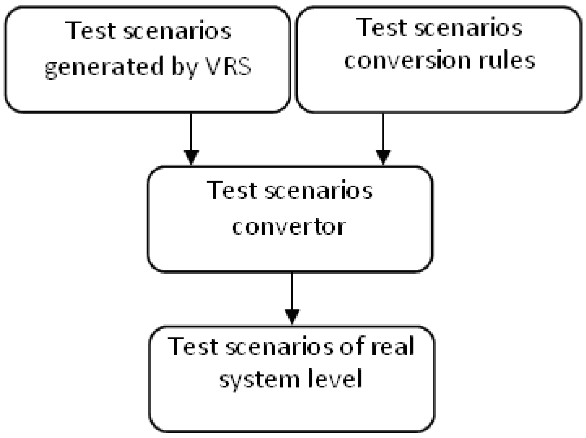

Test scenarios generated by VRS
Test scenarios conversion rules
Test scenarios convertor
Test scenarios of real system level

Fig. 12. Test scenarios conversion scheme

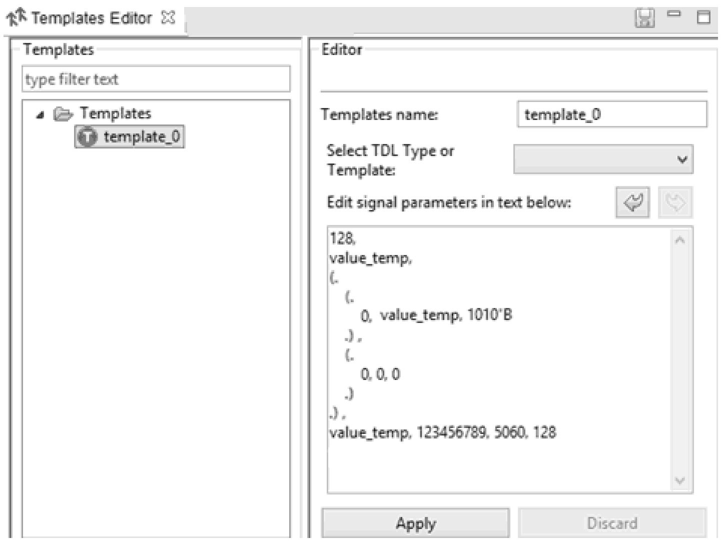
Detailing stage is based on the grammar of data structures conversion rules described in Fig. 8 and conversion algorithm. The specific feature of test automatic scenarios detailing to the level of real system is storing of proved properties of the system obtained in process of abstract model verification.

6. Templates

Often similar conversion rules are required for different signals. Templates can be used to simplify this approach. A developer can define a template of detailed signal, specify either formula or concrete values as a parameter of detailed signal and then apply this template for all required signals. For each case of template usage a developer can specify missed values in the template, change the template itself or modify its structure without violating specified limitations. Templates mechanism simplifies significantly the process of conversion rules creation.

Consider the process of templates usage. Templates are created in separate editor (Templates Editor). In Fig. 13 the template "template_0" is shown which contains detailed data structures inside and the dummy value "value_temp" which shall be changed to concrete values when template is used.

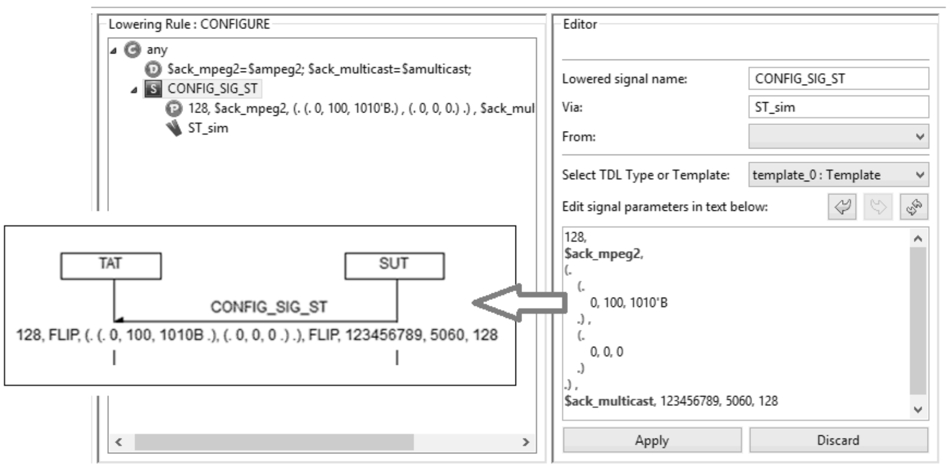
When a template of data structure is ready, it can be used for creation of conversion rules. Fig. 14 represents usage of the template "template_0" with substituted concrete



Templates name: template_0
Select TDL Type or Template:
Edit signal parameters in text below:

```
128,
value_temp,
(.
  (.
    0, value_temp, 1010'B
  .) ,
  (.
    0, 0, 0
  .)
.) ,
value_temp, 123456789, 5060, 128
```

Fig. 13. Example of the template "template_0"



```
$ack_mpeg2=$ampeg2; $ack_multicast=$amulticast;
CONFIG_SIG_ST
128, $ack_mpeg2, (. (. 0, 100, 1010'B.) , (. 0, 0, 0.) .) , $ack_mul
ST_sim
```

128, FLIP, (. (. 0, 100, 1010B .), (. 0, 0, 0 .) .), FLIP, 123456789, 5060, 128

Lowered signal name: CONFIG_SIG_ST
Via: ST_sim
From:
Select TDL Type or Template: template_0 : Template
Edit signal parameters in text below:

```
128,
$ack_mpeg2,
(.
  (.
    0, 100, 1010'B
  .) ,
  (.
    0, 0, 0
  .)
.) ,
$ack_multicast, 123456789, 5060, 128
```

Fig. 14. Applying the template "template_0" for the signal "CONFIG_SIG_ST"

values of signal parameters instead of the dummy value "value_temp" which then will appear in behavioral MSC scenario.

Note that in conversion rules editor complex data structures are represented with formatted text which makes parameters and values more readable than in linear representation of MSC scenario.

Templates usage reduces efforts on creation and coding complex data structures on 25%-30% and reduces possibility of introducing extra bugs because of user inaccuracy.

7. Conclusion

Proposed approach to behavioral scenarios generation based on formal models differs from existing approaches in using the process of automatic conversion of behavioral scenarios with abstract data structures into behavioral scenarios with detailed data structures used in real applications. Proposed language and overall scheme of this process allow automating of creation a set of covering behavioral scenarios. In the scope of this work the analyzer/editor for conversion rules of signals from abstract UCM model level into signals of real system level was developed and called LoweringEditor. It supports the following functionality: automatic binding between conversion rule and signal of UCM level, conversion rules correctness checking, templates usage, highlighting the syntax of conversion rules applying conditions specification, variables usage, libraries and external scripts (includes) usage, splitting UCM signal or action into several signals of real system according to communication protocol, copy/paste/remove operations, import and export from/to storage file. Features described in the article make process of automatic conversion powerful and flexible for different types of telecommunication applications. Adding LoweringEditor into technology process of telecommunication software applications test automation allowed to exclude effort-consuming manual work in the cycle of test suite automated generation for industrial telecommunication applications, increase productivity of test generation in 25% and spread the properties proved on abstract models into generated code of executable test sets. Excluding of manual work allows to reduce human factor in testing process and guaranty quality of generated test suite based on verification results.

References

[1] “Model Driven Architecture - MDA”, *http://www.omg.org/mda*, 2007.

[2] Pastor O. et al., “Model-Driven Development”, *Informatik Spektrum*, **31**:5 (2008), 394–407.

[3] Beydeda S. , Book M., Gruhn V., “Model Driven Software Development”, *Springer-Verlag Berlin Heidelberg*, 2005, 464.

[4] Binder R.V., Kramer A., Legeard B., “2014 Model-based Testing User Survey: Results”, *http://model-based-testing.info/wordpress/wp-content/uploads/2014_MBT_User_Survey_Results.pdf*, 2014.

[5] Fenton N.E., Ohlsson N., “Quantitative analysis of faults and failures in a complex software system”, *Software Engineering, IEEE Transactions on*, 2000, № 8.

[6] Buhr R. J. A., Casselman R. S., “Use Case Maps for Object-Oriented Systems”, *Prentice Hall*, 1995, 302.

[7] Anureev I. et al., “Tools for supporting integrated technology of analysis and verification of specifications for telecommunication applications”, *SPIIRAN works*, **1** (2013), 28.

[8] Letichevsky A.A. et al., “Insertion modeling in distributed system design”, *Problems of programming*, 2008, 13–39.

[9] Kolchin A. et al., “Approach to creating concretized test scenarios within test automation technology for industrial software projects”, *Automatic Control and Computer Sciences, Allerton Press, Inc.*, **47**:7 (2013), 433–442.

DOI: 10.18255/1818-1015-2015-6-750-762

Особенности применения модельно-ориентированного подхода при разработке промышленных приложений

Дробинцев П. Д., Котляров В.П., Воинов Н.В., Никифоров И.В.

получена 21 октября 2015

В статье рассмотрены особенности применения технологий разработки программных систем на основе модельно-ориентированного подхода: Model Driven Software Development (MDSD), Model Driven Architecture (MDA) и Model Driven Development (MDD). Описаны преимущества использования подходов в промышленности. Основной акцент сделан на проектирование систем, автоматическую генерацию кода больших систем, верификацию, доказательство свойств систем и уменьшение плотности ошибок. Приведены недостатки использования данного подхода, одним из которых является различная степень детальности модели и реальной реализованной системы на языке программирования. В работе предлагается подход, характерный для систем, имеющих многоуровневое представление, связанное с детализацией функциональности приложения до уровня, на котором осуществляется прямая генерация корректного кода. Подход позволяет детализировать модель до уровня реального кода системы, при этом сохранить проверенную семантику модели и обеспечить проверку всей детальной модели. Детализация проводится как по потоку управления, так и по потоку данных. Представлены шаги по преобразованию абстрактных структур данных (в том числе транзакций, сигналов и их параметров) в структуры данных, используемых в реализации систем. Приведена грамматика языка задания правил преобразования структур данных абстрактной модели в детальные структуры данных реальной системы и общая схема преобразования. Приведены результаты применения предложенного метода в промышленной технологии.

Статья представляет собой расширенную версию доклада на VI Международном семинаре “Program Semantics, Specification and Verification: Theory and Applications”, Казань, 2015.

Статья публикуется в авторской редакции.

Ключевые слова: модельно-ориентированный подход; многоуровневые модели приложения; спецификация моделей по управлению и структурам данных; верификация моделей; подстановки, сохраняющие корректность доказанных свойств

Для цитирования: Дробинцев П. Д., Котляров В.П., Воинов Н.В., Никифоров И.В., "Особенности применения модельно-ориентированного подхода при разработке промышленных приложений", *Моделирование и анализ информационных систем*, **22**:6 (2015), 750–762.

Об авторах:
Дробинцев Павел Дмитриевич, orcid.org/0000-0003-1116-7765, канд. техн. наук, доцент,
Санкт-Петербургский политехнический университет Петра Великого,
ул. Политехническая, 29, г. Санкт-Петербург, 195251 Россия, e-mail: drob@ics2.ecd.spbstu.ru

Котляров Всеволод Павлович, orcid.org/0000-0003-3973-5218, канд. техн. наук, профессор,
Санкт-Петербургский политехнический университет Петра Великого,
ул. Политехническая, 29, г. Санкт-Петербург, 195251 Россия, e-mail: vpk@spbstu.ru

Воинов Никита Владимирович , orcid.org/0000-0002-0140-1178, канд. техн. наук, доцент,
Санкт-Петербургский политехнический университет Петра Великого,
ул. Политехническая, 29, г. Санкт-Петербург, 195251 Россия, e-mail: voinov@ics2.ecd.spbstu.ru

Никифоров Игорь Валерьевич , orcid.org/0000-0003-0198-1886, канд. техн. наук, доцент,
Санкт-Петербургский политехнический университет Петра Великого,
ул. Политехническая, 29, г. Санкт-Петербург, 195251 Россия, e-mail: igor.nikiforovv@gmail.com

Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 763–772
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 763–772

DOI: 10.18255/1818-1015-2015-6-763-772

UDC 004.054+004.4'23

Fast and Safe Concrete Code Execution for Reinforcing Static Analysis and Verification

Belyaev M., Itsykson V.

Received September 15, 2015

The problem of improving precision of static analysis and verification techniques for C is hard due to simplification assumptions these techniques make about the code model. We present a novel approach to improving precision by executing the code model in a controlled environment that captures program errors and contract violations in a memory and time efficient way. We implemented this approach as an executor module `Tassadar` as a part of bounded model checker `Borealis`. We tested `Tassadar` on two test sets, showing that its impact on performance of `Borealis` is minimal.

The article is published in the authors' wording.

Keywords: concrete interpretation, symbolic execution, static code analysis, analysis precision

For citation: Belyaev M., Itsykson V., "Fast and Safe Concrete Code Execution for Reinforcing Static Analysis and Verification", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 763–772.

On the authors:
Belyaev Mikhail, orcid.org/0000-0003-1260-9211, assistant,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya street, 21, Saint-Petersburg, 194021, Russia
e-mail: belyaev@kspt.icc.spbstu.ru

Itsykson Vladimir, orcid.org/0000-0003-0276-4517, PhD,
Peter the Great St. Petersburg Polytechnic University,
Polytechnicheskaya street, 21, Saint-Petersburg, 194021, Russia
e-mail: vlad@icc.spbstu.ru

Introduction

Static analysis and verification of programs written in an unsafe language like C is hard. Most of the problems researchers face in these areas are either NP-hard or undecidable due to inherent properties of a Turing-complete language and the presence of unsafe memory operations. Another difficulty comes from the fact that the analysis that can be used in an interprocedural environment must be aware of both internal and external functions to provide a good approximation of program behaviour.

The goal of this work is to provide a safe and fast way to reduce the number of false positives in results produced by static code analysis of C. We mainly focus on logic-based analysis techniques (such as bounded model checking [5]) that allow to find non-functional program defects (i.e. null pointer deferences, buffer overflows, division by zero, etc.) and code contract violations. We provide a way to reduce false positives in these cases in the form of a concrete code execution environment that borrows some ideas from symbolic execution and is safe, robust and resource-efficient.

This technique has been implemented in a prototype executor called `Tassadar` as a module of a bounded model checker `Borealis` [1] and was tested using `Borealis` testbench.

The rest of this paper is structured as follows. The problem statement is given in section 1. Section 2 is dedicated to describing the execution environment itself. The implementation details are given in section 3. Section 4 includes our evaluation results. Related work and summary of alternate approaches are given in section 5.

1. Problem Statement

Most approaches to finding program defects and/or checking source contracts have limitations that result in lowering code model conformance as a trade-off between the quality of analysis and resource consumption. The most basic of these limitations are as follows.

1. Limiting the number of loop iterations and recursive function calls;
2. Simplified view on memory and pointer aliasing;
3. Using summaries and/or contracts to model function calls;
4. Replacing stdlib/system/external function calls with annotations or approximated models.

For the rest of this paper we will use the term *analysis* (or *the* analysis) to refer to the static analysis or verification technique that is being augmented by our approach and assume that the aim of this analysis is to either detect program defects or contract violations, referred to as *errors*.

There are two basic measures for analysis quality: *precision* and *recall* [6]. Precision and recall are dual properties that cannot usually be achieved together at the same time: any improvement to recall usually raises the number of false positives as well, lowering precision, and any change that increases precision usually lowers the number of true positives, having a negative impact on recall. **The idea of this work** is to improve analysis precision by filtering out false positives *after* the analysis has finished, thus having no impact on recall. Improving precision is very important for real-life usage of defect detection tools as false positives make analysis results noisy and less usable.

We do this by using a technique that is based on directly executing all program procedures in a checked and analysis-aware environment. The set of properties this process must satisfy is as follows.

- Avoid infinite and lengthy execution;
- Minimize resource consumption to the lowest level possible;
- Avoid as many of model inaccuracies imposed by the analysis as possible;
- Capture all analysis-supported errors.

We must also keep notice that some values (e.g., a call to `rand()` that results in a defect only for some return values) must be handled in an analysis-conformant way, returning the same value that was inferred by the analysis if possible. This is also true for initial function arguments and global variables for a concrete function execution. Without this, it would be impossible to assess whether the analysis-provided result was correct.

2. Code Execution in a Controlled Environment

This section details our approach, including dealing with standard register operations, memory simulation and calling external libraries. We also provide some details on how we deal with code contracts.

2.1. Modeling register operations

In this paper we assume that the code model conforms to the static single assignment (SSA [8]) form. This is a widely used code model (e.g., used by GCC [14] and LLVM [13] infrastructures). This provides us with ability to work on already optimized code for everything besides memory operations. Implementing numeric operations is pretty straightforward using biggest numeric types available. All the work with structures/unions is done via memory.

Calling internal functions is done through modeling call stack and current instruction pointer. As all the values in SSA form are assigned only once, we can store their values in a flat value table. We also keep track of stack-allocated pointers for deallocation. In order to conform to the variability of results provided by analysis techniques, some values in the code should be treated as unknown, or *symbolic*. The biggest difference between our approach and symbolic execution is that we always try to get concrete values for all these symbols using a special facility called *the arbiter*.

Arbiter is a function from symbols to real values that has external information about the code that the executor have no access to. When checking analysis-based counterexamples, it is the arbiter that provides the connection between the executor and the analysis. Each query for a value that cannot be directly computed results in a mapping of the corresponding variable from the counterexample. However, the arbiter interface is independent from the analysis itself, providing a point of extension.

A special case of symbolic values are pointers. All the symbolic pointers are modeled using a special non-null value that is known both to arbiter and memory model. Dereferencing this value effectively queries the arbiter instead of the memory.

Currently we do not support writing values to symbolic pointers and this presents a problem when dealing with structures passed by pointer to top-level functions, but this problem can be dealt with by reconstructing the storage graph from the counterexample, which is possible but complex and not yet implemented in our prototype.

2.2. Modeling memory operations

Segment tree (ST) is a well-known data structure first introduced by [3]. It is essentially a balanced tree with leaves representing elements and inner nodes representing ranges, the root of the tree being the whole range and every child representing a range that is one half of its parent's. This data structure can be used to efficiently perform a number of tasks, like the minimum range query or any other range query based on an associative binary operation. Changing a leaf value and recalculating the whole tree is a $O(log_2(N))$ operation for a range of size N.

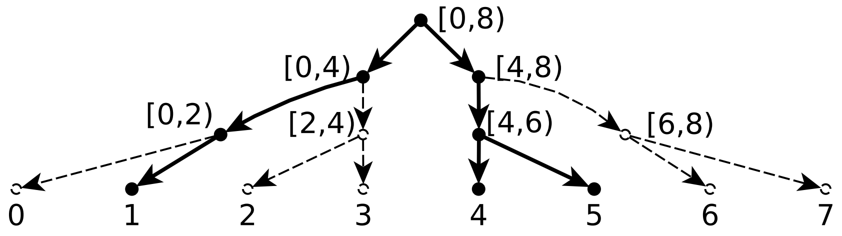


Fig. 1. Implicit segment tree

This structure is very inefficient memory-wise. This can be countered by using an *implicit* version of segment tree, where tree nodes are constructed during writes only. Figure 1 shows this difference in detail: an ST needs 15 cells for 8 elements, while an implicit ST (shown in bold) only 7 cells for 3 writes. This change imposes no penalty on lookup complexity. Memory consumption becomes $O(Klog_2(N))$ where K is the number of write queries. The difference can be dramatic for big arrays. E.g., for a range of size 2^{31} building an ST will require 4294967294 cells, while for the implicit version it depends on the number of writes, which is much smaller. Another optimization is to reduce height by storing fixed size flat arrays in leaves, which reduces the height by $log_2(P)$ but adds $O(P)$ to the complexity of each write.

As mentioned earier, an implicit ST can be efficiently used to represent big amounts of data without consuming too many resources. In order to provide all the desired memory properties for our executor, like handling buffer overflows, allocations, deallocations and so on, we need to store this cumulative information in the nodes themselves in a manner similar to the one used when solving range queries.

Each node in the tree contains a tuple $\{Q, S, F, Sz, D\}$. Memory status Q (`Alloca`, `Malloc` or `Unknown`) designates whether this exact node was used as the root of an allocated memory region with size Sz. Memory state S (`Memset`, `Uninitialized` or `Unknown`) is used to represent zero-initialized, uninitialized or `memset` region of memory, enabling shortcut reads for regions that are allocated, but never written to. The byte F represents the byte value for `memset`.

Each ST leaf contains a pointer to buffer containing the memory chunk and does not contain child pointers. The usual memory access pattern in C implies that the size of chunks stored in leaves should be at least as big as structure padding and preferably a multiple of it. With exception to direct memory access functions and unpadded structures, this will ensure that every read or write accesses one chunk only.

Summing up, every node in the tree represents a memory range of size $2^N K$ where K is the size of leaf memory chunk. To allocate a new memory range we need to find a node representing a big enough range and set Q and S accordingly. No nodes below this level are created until written to.

To perform a write we descend down the tree until we reach the desired chunk, splitting the data source array and creating new nodes if needed. An important thing is keeping track of the value of S. For each new child created, we create its sibling, copy S and F from parent and reset S in parent to unknown. A read is the same, but does not create new nodes and merges the data array on the way back up. If we encounter a node with $S = \texttt{Memset}$, the returned value is set to F.

Other primitive operations are `memset` (different to write in that it assigns S and F for appropriate ranges) and `memchr` (similar to read except that it traverses the tree horisontally) because there is no way to do them efficiently through reads and writes.

On each tree traversal a *single allocated* node on the way down is searched for and bound-checked. Any situation when this is violated is an error.

2.3. Modeling global state

Global variables and constants have a much simpler structure. All values in this space have known size and need no additional checks for initialization [10]. In order to keep track of global variables, we use a standard binary search tree of pointers to lazily-allocated buffers. In the future, it might be more convenient to use a unified structure for both local and global data. Pointers to labels and functions are stored separately in a structure associating them with real pointers. Doing arithmetics on these pointers is an error.

The only query needed for global data is to be able to find the base pointer for a value. This is done using a standard binary search. Local and global memories are distinguished by the ranges of corresponding pointers.

2.4. Handling external functions

We do support all the functions of the standard C library and the most common POSIX functions. The system-based side effects, like networking, files, etc. are not simulated, but the values produced are sensible for their counterparts, including error situations. Functions without side effects (for example, the builtin math library) are implemented directly.

Functions directly operating the memory are implemented on top of primitive reads, writes, `memset` and `memchr`. Another function that is subject to become primitive is memcpy, which is rather inefficient at the moment and making it efficient is a subject of further research.

Another important thing is support for external libraries, which at the moment can only be implemented as a part of the executor itself. Providing an external interface to implement these is also a subject of future research.

2.5. Checking code contracts

Most programming tools that support contracts either specify them inside the language (implementing these is no different from other functions) or outside, be it comments, annotations or external data files. These definitions must be parsed separately and are usually provided as their AST. As these contracts are (usually) not allowed to have any side effects, thus executing them is straightforward using the AST itself.

Some systems (notably the ACSL annotation language used in Frama-C [2]) provide a framework for specifying contract in a more logic-oriented way, supporting quantifiers over logic formulae, providing user-created theorems and other constructs that are more typical for logic programming languages rather than functional or imperative ones and cannot be in any way *executed* in the sense we put to this term. The reason for this is that "executing" a quantified formula effectively means checking it for every possible value of all bound variables, which violates our goals and is impossible in the general case. However, in some simple cases these can be reduced to non-quantified variants by means of skolemization (or hebrandization) of these formulae. In other cases, the set of values for each bound variable is known at runtime and they can be checked iteratively. At the moment, such contracts are ignored. Approximating checks with these contracts in a reasonable way is a subject for future research.

3. Implementation details

We have implemented our approach in a module called Tassadar in a bounded model checker tool Borealis [1] using the infrastructure borrowed from LLVM interpreter library. Borealis is based on LLVM IR and supports several ways of specifying code contracts. The LLVM interpreter is a simple instruction interpreter that operates on LLVM IR instructions and in order to implement the checks needed by our tool the whole memory model, external function calls and some parts of regular instruction handling had to be written from scratch. We also implemented executing and checking Borealis contract specifications.

Borealis is based on using an SMT-solver for checking desired properties. Every found defect or contract violation provides a counterexample — a set of values that could lead to an erroneous situation at run-time. We re-package the values from this counterexample to a hashtable and use that as the basis for our analysis-driven arbiter. The whole solution is implemented in C++ and packaged as a set of LLVM passes that are run by Borealis.

4. Evaluation

We evaluated our approach and `Tassadar` on two test case sets bundled with `Borealis` which are based on NECLA [11] and SV-COMP [4] test case packs, which test both defect detection and contract violation detection properties of the checker. This includes 84 testcases with over 40 defects/violations, about 30% of which are false positives in the first set and 20 test cases with around 12 defects/violations with 30% false positive ratio in the second set. The main difference is that the second set contains much more complex programs with heavy memory usage. `Tassadar` was required to check every positive detected by `Borealis`. In total it has identified 16 false positives in both test sets and no false negatives.

Table 1. Test run results summary

	Time+, s	Time−, s	Time%	Mem+,	Mem−,	Mem%	FPs
Set 1	19.017	18.976	0.2%	71366	70189	17%	11
Set 2	107.844	107.678	0.1%	173553	173697	0.08%	5

The test runs were performed on a AMD Phenom(tm) II X4 machine with 4 cores and 8 Gbs of RAM. The test bench was analyzed 10 times with `Tassadar` and without it. The results of the runs are summarized in table 1. Time+ and Time− is time spend with executor and without it, Mem+ and Mem− is the top memory consumption for both. The "%" columns show the approximate percentage of corresponding resource spent by `Tassadar`. The main goal of this work is to provide a result checker that has minimal impact compared to the run time of the analysis. As one can see from the table, this impact does never exceed 1 percent of total run time.

5. Related work

The techniques based on augmenting analysis techniques through with "simulation" of produced counterexamples is well-known as the basis for abstraction refinement-based approaches, e.g. for traditional model checking [7]. It was later extended by [12] to actually *execute* the code to provide information suitable for refinement. This and later workings (for example, [9]) propose implementing abstraction refinement through code-to-code instrumentation, compiling and running instrumented code using traditional means.

This is quite similar to our approach, albeit for a different purpose. A similar technique could be implemented for our problem, but using instrumented code execution limits the ability to reason about inner state of the program. None of these papers actually give a description of data structures used to capture, store and query memory range information, but it is hinted at that they use a simple linear container of ranges in addition to the memory itself, which has linear time and memory complexity over the number of ranges. The program itself should have means of communication (through replacing different external functions) with the checker during runtime. This approach is also not capable of capturing some buffer overflows and illegal pointer dereference when illegal pointers accidentally point to legal data.

[9] provide evaluation results for their approach and state the total execution overhead

as 1% to 12% of all runtime, which is much worse than our approach gets, but they are not directly comparable due to different analysis techniques used.

Conclusion

This paper is focused around building an efficient executor of C code for reinforcing results of static analysis using dynamic result checks. We build our approach around creating an interpreter that operates on SSA-based compiler model that allows us to use the compiler-based standard optimizations and provides a controlled environment that is able to capture and check all the analyzed properties at the same time.

We have built a prototype implementation on top of the LLVM compiler system and `Borealis` bounded model checker that has proven to be very resource-efficient with comparison to the checker itself and, at the same time, being able to correctly identify a portion of false positives.

In the future we plan to apply this executor to improving the analysis quality further, building a CEGAR-like refinement loop on top of it, support the things we do not currently support (fully rebuild top function arguments passed by pointer, use an external language for external functions, explore the possibilities of checking quantifier-based and intrinsic contracts, etc.) and do a more thorough comparison with other similar tools.

We also plan to research the possibilities of applying the existing implementation to different areas it could be used in, like checking and reducing tests produced by test generation (of which `Borealis` has limited support [?]), checking results for other kinds of analysis, explore the possibility for dynamic analysis in general to check arbitrary code properties during execution. It is also possible to try building a memory-efficient time-travelling interpreter on top of `Tassadar` as the data structure used to model the memory is persistent and implementing versioning on top of it is pretty straightforward.

References

[1] M. Akhin, M. Belyaev, V. Itsykson, “Software defect detection by combining bounded model checking and approximations of functions”, *Automatic Control and Computer Sciences*, **48**:7 (2014), 389–397.

[2] P. Baudin, J. C. Filliâtre, T. Hubert, C. Marché, B. Monate, Y. Moy, V. Prevosto, 2008, ACSL: ANSI/ISO C Specification Language. Preliminary Design, version 1.4., Preliminary.

[3] J. L. Bentley, “Solutions to klee’s rectangle problems”, Technical report, 1977.

[4] D. Beyer, “Competition on software verification”, 2012, 504–524.

[5] A. Biere, A. Cimatti, E. M. Clarke, Y. Zhu, “Symbolic model checking without BDDs”, 1999, 193–207.

[6] M. K. Buckland, F. C. Gey, “The relationship between recall and precision”, *JASIS*, **45**:1 (1994), 12–19.

[7] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, “Counterexample-guided abstraction refinement”, *CAV*, 2000, 154–169.

[8] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, F. K. Zadeck, “Efficiently computing static single assignment form and the control dependence graph”, *ACM TOPLAS*, **13**:4 (1991), 451–490.

[9] A. Groce, R. Joshi, “Extending model checking with dynamic analysis”, *Verification, model checking, and abstract interpretation*, 2008, 142–156.

[10] ISO, The ANSI C standard (C99), ISO/IEC, 1999.

[11] F. Ivančić, S. Sankaranarayanan, “NECLA static analysis benchmarks”, 2009.

[12] D. Kroening, A. Groce, E. Clarke, “Counterexample guided abstraction refinement via program execution”, *Formal methods and software engineering*, 2004, 224–238.

[13] C. Lattner, V. Adve, “LLVM: A compilation framework for lifelong program analysis & transformation”, 2004, 75–86.

[14] D. Novillo, “Tree SSA: A new optimization infrastructure for GCC”, *Proceedings of the 2003 gCC developers' summit*, **21**, 2014, 83–93.

DOI: 10.18255/1818-1015-2015-6-763-772

Эффективное исполнение программного кода в контролируемом окружении как способ улучшения результатов статического анализа и верификации программ

Беляев М.А., Ицыксон В.М.

получена 15 сентября 2015

Существующие средства и методы статического анализа и верификации кода на языке C используют различные методики упрощения программной модели, приводящие к значительному снижению точности анализа. В данной работе представлен новый подход к повышению точности анализа путем исполнения программной модели в контролируемом окружении, которое позволяет точно определять ошибочные ситуации, такие, как нарушения контрактов кода и ошибки работы с памятью, оставаясь при этом эффективным с точки зрения затрат по времени и по памяти. Данный подход был реализован в модуле под названием «Tassadar» в рамках средства ограниченной проверки моделей «Borealis». Прототип был опробован на стандартных наборах тестовых программ данного средства и показал минимальное влияние на его общую производительность.

Статья представляет собой расширенную версию доклада на VI Международном семинаре “Program Semantics, Specification and Verification: Theory and Applications”, Казань, 2015.

Статья публикуется в авторской редакции.

Ключевые слова: программная интерпретация, символьное исполнение, статический анализ программного кода, точность анализа

Для цитирования: Беляев М.А., Ицыксон В.М., "Эффективное исполнение программного кода в контролируемом окружении как способ улучшения результатов статического анализа и верификации программ", *Моделирование и анализ информационных систем*, **22**:6 (2015), 763–772.

Об авторах:
Беляев Михаил Анатольевич, orcid.org/0000-0003-1260-9211, ассистент
Санкт-Петербургский политехнический университет им. Петра Великого,
194021, Россия, г. Санкт-Петербург, Политехническая ул., 21
e-mail: belyaev@kspt.icc.spbstu.ru

Ицыксон Владимир Михайлович, orcid.org/0000-0003-0276-4517, доцент
Санкт-Петербургский политехнический университет им. Петра Великого,
194021, Россия, г. Санкт-Петербург, Политехническая ул., 21
e-mail: vlad@icc.spbstu.ru

Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 773–782
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 773–782

DOI: 10.18255/1818-1015-2015-6-773-782

UDC 519.681.3

Loop Invariants Elimination for Definite Iterations over Unchangeable Data Structures in C Programs

Maryasov I. V.[1], Nepomniaschy V. A.

Received October 26, 2015

The C-program verification is an urgent problem of modern programming. To apply known methods of deductive verification it is necessary to provide loop invariants which might be a challenge in many cases. In this paper we consider the C-light language [18] which is a powerful subset of the ISO C language. To verify C-light programs the two-level approach [19, 20] and the mixed axiomatic semantics method [1, 3, 11] were suggested. At the first stage, we translate [17] the source C-light program into C-kernel one. The C-kernel language [19] is a subset of C-light. The theorem of translation correctness was proved in [10, 11]. The C-kernel has less statements with respect to the C-light, this allows to decrease the number of inference rules of axiomatic semantics during its development. At the second stage of this approach, the verification conditions are generated by applying the rules of mixed axiomatic semantics [10, 11] which could contain several rules for the same program statement. In such cases the inference rules are applied depending on the context. Let us note that application of the mixed axiomatic semantics allows to significantly simplify verification conditions in many cases. This article represents an extension of this approach which includes our verification method for definite iteration over unchangeable data structures without loop exit in C-light programs. The method contains a new inference rule for the deifinite iteration without invariants. This rule was implemented in verification conditions generator. At the proof stage the SMT-solver Z3 [12] is used. An example which illustrates the application of this technique is considered.

The article is published in the authors' wording.

Keywords: C-light, loop invariants, mixed axiomatic semantics, definite iteration, unchangeable data structures, Z3, specification, verification, Hoare logic

For citation: 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.

On the authors:
Maryasov Ilya Vladimirovich, orcid.org/0000-0002-2497-6484, PhD,
A.P. Ershov Institute of Informatics Systems SB RAS
Akademik Lavrentiev pr., 6, Novosibirsk, 630090, Russia,
e-mail: ivm@iis.nsk.su

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

Acknowledgments:
[1]This research is partially supported by RFBR grant 15-01-05974.

Introduction

C program verification is an urgent problem at the present time. Many projects (for example [4, 5, 6, 8, 9]) suggest different solutions. But none of them contains any methods for loop verification. As it is known, in order to verify loops we need invariants whose construction is a challenge. So the user has to devise these invariants. In many cases it is a difficult task.

In this paper we suggest a method of loop invariants elimination for definite iteration of special form [14]. We extend our mixed axiomatic semantics of C-light language by a new rule which allows verification of such loops without invariants provided by user.

C-light language [18] is a powerful subset of the C language. To verify C-light programs the two-level approach [19, 20] and the mixed axiomatic semantics method [1, 11] were suggested.

On the first stage, we translate [17] the source C-light program into C-kernel one. C-kernel language [19] is a subset of C-light. On the second stage, the verification conditions are generated by applying the rules of mixed axiomatic semantics [10, 11]. The word "mixed" means that it can be several inference rules for the same program construction which are unambiguously applied depending on its context. In many cases the use of specialized inference rules allows us to simplify verification conditions.

All our methods have theoretical justification. Theorems of correctness of translation of C-light into C-kernel and soundness of the C-kernel axiomatic semantics are proven in [17, 10].

At the proof stage, the automatic theorem prover Z3 [12] is used. Extra axioms can be provided by the user in case the prover has failed to check whether a verification condition is true. If all verification conditions have been proven, then the program is partially correct. Otherwise, the user has to modify the program or its specification and to repeat the verification process in C-light verification system [11, 16].

1. Definite Iteration over Unchangeable Data Structures and Replacement Operation

The method of loop invariants elimination for definite iteration was suggested in [14]. It includes four cases [13, 15]:

1. Definite iteration over unchangeable data structures without loop exit.

2. Definite iteration over unchangeable data structures with loop exit.

3. Definite iteration over changeable data structures possibly with loop exit.

4. Definite iteration over hierarchical data structures possibly with loop exit.

This paper deals with the first case.

Let us remind the notion of data structures which contain a finite number of elements. Let $memb(S)$ be the multiset of elements of the structure S and $|memb(S)|$ be the power of the multiset $memb(S)$. For the structure S the following operations are defined:

1. $empty(S) = true$ iff $|memb(S)| = 0$.
2. $choo(S)$ returns an element of $memb(S)$ if $\neg empty(S)$.
3. $rest(S) = S'$, where S' is a structure of the type of S and $memb(S') = memb(S) \setminus \{choo(S)\}$ if $\neg empty(S)$.

Sets, sequences, lists, strings, arrays, files and trees are typical examples of the data structures.

Let $\neg empty(S)$, then $vec(S) = [s_1, s_2, \ldots, s_n]$ where $memb(S) = \{s_1, s_2, \ldots, s_n\}$ and $s_i = choo(rest^{i-1}(S))$ for $i = 1, 2, \ldots, n$.

$last(S)$ is a partial function such that $last(S) = s_n$.

A function $head(S)$ returns a structure such that $vec(head(S)) = [s_1, s_2, \ldots, s_{n-1}]$ if $\neg empty(S)$.

Let S_1 and S_2 be structures. Then we can define a concatenation operation $con(S_1, S_2)$ as follows:

1. $con(S_1, S_2) = S_2$ if $empty(S_1)$.
2. $choo(con(S_1, S_2)) = choo(S_1)$ and $rest(con(S_1, S_2)) = con(rest(S_1), S_2)$ if $\neg empty(S_1)$.

Consider the statement

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

where S is a structure, x is the variable of the type of S element, v is the vector of loop variables which does not contain x and $body$ represents the loop body computation, does not modify x and S and terminates for each $x \in memb(S)$. The loop body can contain only the assignment statements and the *if* statements, possibly nested.

The operational semantics of such statement is defined as follows. Let v_0 be the vector of initial values of variables from v. If $empty(S)$ then the result of the iteration $v = v_0$. Otherwise, if $vec(S) = [s_1, s_2, \ldots, s_n]$, then the loop body iterates sequentially for x taking the values $s_1, s_2, \ldots, s_n$.

To express the effect of the iteration let us define a replacement operation

$$rep(v, S, body) = v_n,$$

where $v_0 = v$ if $empty(S)$, $v_i = body(v_{i-1}, s_i)$ for all $i = 1, 2, \ldots, n$ if $\neg empty(S)$.

A number of theorems which express important properties of the replacement operation was proved in [13, 14, 15]. Let us mention the most important of them.

Theorem 1. $rep(v, con(S_1, S_2), body) = rep(rep(v, S_1, body), S_2, body)$.

Theorem 2. $\neg empty(S) \Rightarrow rep(v, S, body) = body(rep(v, head(S), body), last(S))$.

2. The Inference Rule and Its Implementation

There is no *for* statement in C-kernel. The loop **for** $(\mathbf{e_1}; \mathbf{e_2}; \mathbf{e_3})\ \mathbf{B}$; is translated first into the *while* statement $\mathbf{e_1}$; **while** $(\mathbf{e_2})\ \{\mathbf{B}; \mathbf{e3};\}$; and then the common inference rule of the mixed axiomatic semantics is applied:

$$\frac{\begin{array}{c} E, SP \vdash \{P\}\, \mathbf{e_1};\{INV\} \\ E, SP \vdash \{INV \wedge \mathit{cast}(\mathit{val}(e_2, MD), \mathit{type}(e_2), \mathit{int}) \neq 0\}\, \mathbf{B}; \mathbf{e_3};\{INV\} \\ E, SP \vdash \{INV \wedge \mathit{cast}(\mathit{val}(e_2, MD), \mathit{type}(e_2), \mathit{int}) = 0\}\, \mathbf{A};\{Q\} \end{array}}{E, SP \vdash \{P\}\, \mathbf{e_1};\, \{INV\}\, \mathbf{while}\ (\mathbf{e_2})\ \{\mathbf{B}; \mathbf{e_3}\}\, \mathbf{A};\{Q\}}$$

Here P is precondition, Q is postcondition, INV stands for loop invariant, $\mathbf{A}$ are program statements after the loop.

E is the environment [10] which contains an information about current function (its identifier, type and body) which is verified, an information about current block and label identifier if **goto** statement occurred earlier.

SP is program specification which includes all preconditions, postconditions and invariants of loops and labeled statements.

The function *cast* performs type casting according to ISO C standard, the function *val* calculates the value of the expression e_2, the function *type* returns the type of e.

The meta-variable MD defines the values stored in memory [1, 10].

Now we can introduce the special inference rule for definite iterations:

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

We use forward tracing: we move from the program beginning to its end and eliminate the leftmost operator (on the top level) applying the corresponding rule. The correctness of this rule can be proved by modification of the proof for backward tracing from [14].

The implementation of this rule extends our verification conditions generator which is based on mixed axiomatic semantics of C-kernel [1, 10].

Note that the common rule adds at least two verification conditions and the rule for definite iteration does not increase the number of verification conditions for a program.

In C-light there is no such statement as $\mathbf{for\ x\ in\ S\ do\ v} := \mathbf{body}(\mathbf{v}, \mathbf{x})\ \mathbf{end}$. So in fact the generator of verification conditions must be able to determine x, S, v and *body* in a loop of a form $\mathbf{for}\,(\mathbf{e_1}; \mathbf{e_2}; \mathbf{e_3})\, \mathbf{B};$.

Depending on the data structure S we have to introduce several inference rules for each case. For example if S is a subset of integers we have the following rule:

$$\frac{E, SP \vdash \{\exists v'\ P(v \leftarrow v')\ \wedge v = \mathit{rep}(v', (j, j + c, j + 2c, \ldots), \mathit{body})\}\, \mathbf{A};\{Q\}}{E, SP \vdash \{P\}\, \mathbf{for}\ (\mathbf{i} = \mathbf{j}; \mathbf{i} < \mathbf{k}; \mathbf{i} = \mathbf{i} + \mathbf{c};\,)\ \mathbf{v} = \mathbf{body}(\mathbf{v}, \mathbf{i});\ \mathbf{A}\{Q\}}$$

Here i, j, k, c are integers. In the case when $i \geq k$ or $i = i - c$ the rule looks similarly.

Every time when there is no invariant provided by the user before the for statement the generator of verification conditions tries to apply one of the inference rules suggested by our method. When it fails an error is raised and the user has to provide an invariant himself.

3. Example

To demonstrate the application of our method of loop invariants elimination let us consider the following program. It iterates over an array of integers and for given integer computes the number of entries to this array.

The annotated (in SMT-LIB v2 syntax of Z3) C-kernel program has the form:

```
/* (assert (> length 0)) */
int count(int key, int* arr, int length)
{
    auto int result = 0;
    for (i = 0; i < length; i = i + 1)
         if (arr[i] == key) result = result + 1;
    return result;
}
/* (assert (= result (COUNT key a 0 (- length 1)))) */
```

The function *COUNT* returns the number of entries of *key* to *arr* from *arr*[*j*] to *arr*[*k*]. It is defined recursively as follows:

```
(declare-fun COUNT (Int (Array Int Int) Int Int) Int)

(assert
       (and
           (forall ((j Int) (k Int))
                  (implies
                         (> j k)
                         (= (COUNT key arr j k) 0)
                  )
              )
           (forall ((j Int) (k Int))
                  (implies
                         (and
                             (= j k)
                             (= (select arr k) key)
                         )
                         (= (COUNT key arr j k) 1)
                  )
           )
           (forall ((j Int) (k Int))
                  (implies
                         (and
                             (= j k)
                             (not
                                 (= (select arr k) key)
                             )
                         )
                         (= (COUNT key arr j k) 0)
                  )
           )
           (forall ((j Int) (k Int))
                  (implies
                         (and
                             (< j k)
```

```
                            (= (select arr k) key)
                        )
                        (=
                          (COUNT key arr j k)
                          (+ (COUNT key arr j (- k 1)) 1)
                        )
                )
            )
            (forall ((j Int) (k Int))
                (implies
                        (and
                            (< j k)
                            (not (= (select arr k) key))
                        )
                        (=
                          (COUNT key arr j k)
                          (COUNT key arr j (- k 1))
                        )
                )
            )
        )
)
```

The first conjunct describes the case when the second bound is greater then the first bound. The second and the third conjuncts defines the behavior of $COUNT$ when the length of arr is equal to 1. The fourth conjunct increases the value of $COUNT$ by 1 in the case when the entry of key was found at the index k. Otherwise $COUNT$ is equal to $COUNT$ from j to $k-1$ as it is defined by the last conjunct.

Z3 is the SMT-solver but we are interested in verification conditions validity, not satisfiability. So the verification conditions generator produces the negation of the verification condition:

```
(assert
    (not (forall ((key Int) (arr (Array Int Int)) (length Int))
                (implies
                        (exists ((result!1 Int))
                                (and
                                    (> length 0)
                                    (= result!1 0)
                                    (= result (rep result arr length))
                                )
                        )
                        (= result (COUNT key arr 0 (- length 1)))
                )
          )
    )
)
```

And then we expect the answer “unsat” which means that the negation is unsatisfiable so the verification condition is true.

Also the generator produces the recursive definition of the *rep* function for this program:

```
(declare-fun rep (Int (Array Int Int) Int) Int)

(assert (and (forall ((i Int))
                  (implies
                          (< i 0)
                          (= (rep result arr i) 0)
                  )
             )
           (forall ((i Int))
                  (implies
                          (= i 0)
                          (= (rep result arr i) 0)
                  )
           )
           (forall ((i Int))
                  (implies
                          (and
                              (< 0 i)
                              (= (select arr (- i 1)) key)
                          )
                          (=
                            (rep result arr i)
                            (+ (rep result arr (- i 1)) 1)
                          )
                  )
           )
           (forall ((i Int))
                  (implies
                          (and
                              (< 0 i)
                              (not
                                  (= (select arr (- i 1)) key)
                              )
                          )
                          (=
                            (rep result arr i)
                            (rep result arr (- i 1))
                          )
                  )
           )
        )
)
```

Unfortunately Z3 does not support proofs by induction. In this example it goes into infinite loop without any answer. During our experiment we substituted constants for $length$ in the verification condition, and it turned out that for example for $length = 15$ it took 70 seconds for Z3 to provide the desired answer "unsat" running on AMD Athlon II X2 245 processor at 2.9 GHz with 4 gigabytes of RAM.

4. Conclusion

This paper represents an extension of the system for C-light program verification. In the case of definite iteration over unchangeable data structures without loop exit this extension allows to generate verification conditions without loop invariants.

This generation is based on the new inference rule for the C-light for statement which introduces the replacement operation. It expresses definite iteration in special form.

K. Rustan M. Leino suggested a rewriting strategy and a heuristic for when to apply it to verify simple inductive theorems [7]. We plan to use this tactic in our generator of verification conditions.

The next step will be the case of loop invariants elimination for changeable data structures possibly with loop exit.

References

[1] I. S. Anureev, I. V. Maryasov, V. A. Nepomniaschy, "C-programs Verification Based on Mixed Axiomatic Semantics", *Automatic Control and Computer Sciences*, **45**:7 (2011), 485–500, http://link.springer.com/article/10.3103/S0146411611070029.

[2] I. Anureev, I. Maryasov, V. Nepomniaschy, "Revised Mixed Axiomatic Semantics Method of C Program Verification", *Proceedings*, Third Workshop "Program Semantics, Specification and Verification: Theory and Applications", PSSV 2012 (Nizhni Novgorod, Russia, July 1–2), eds. Valery Nepomniaschy, Valery Sokolov, 2012, 16–23.

[3] I. S. Anureev, I. V. Maryasov, V. A. Nepomniaschy, "Two-level Mixed Verification Method of C-light Programs in Terms of Safety Logic", Computer Science, **34**, NCC Publisher, 2012, 23–42.

[4] M. Barnett, B.-Y. E. Chang, R. DeLine, B. Jacobs, K. Rustan M. Leino, "Boogie: A Modular Reusable Verifier for Object-Oriented Programs", *Formal Methods for Components and Objects*, 4th International Symposium, FMCO 2005 (Amsterdam, The Netherlands, November 1-4), Lecture Notes in Computer Science, **4111**, Springer, 2006, 364–387, http://link.springer.com/chapter/10.1007/11804192_17.

[5] E. Cohen, M. Dahlweid, M. Hillebrand, D. Leinenbach, M. Moskal, T. Santen, W. Schulte, S. Tobies, "VCC: A Practical System for Verifying Concurrent C", *Theorem Proving in Higher Order Logics*, 22nd International Conference, TPHOLs 2009 (Munich, Germany, August 17–20), Lecture Notes in Computer Science, **5674**, Springer, 2009, 23–42, http://link.springer.com/chapter/10.1007/978-3-642-03359-9_2.

[6] J.-C. Filliâtre, C. Marché, "Multi-prover Verification of C Programs", *Formal Methods and Software Engineering*, 6th International Conference on Formal Engineering Methods, ICFEM 2004 (Seattle, WA, USA, November 8–12), Lecture Notes in Computer Science, **3308**, Springer, 2004, 15–29, http://link.springer.com/chapter/10.1007/978-3-540-30482-1_10.

[7] K. Rustan M. Leino, "Automating Induction with an SMT Solver", *Verification, Model Checking, and Abstract Interpretation*, 13th International Conference, VMCAI 2012 (Philadelphia, PA, USA, January 22–24), Lecture Notes in Computer Science, **7148**, Springer, 2012, 315–331, http://link.springer.com/chapter/10.1007/978-3-642-27940-9_21.

[8] K. Rustan M. Leino, "Dafny: An Automatic Program Verifier for Functional Correctness", *Logic for Programming, Artificial Intelligence, and Reasoning*, 16th International Conference, LPAR-16 (Dakar, Senegal, April 25–May 1), Lecture Notes in Computer Science, **6355**, Springer, 2010, 348–370, http://link.springer.com/chapter/10.1007/978-3-642-17511-4_20.

[9] X. Leroy, "Formal Verification of a Realistic Compiler", *Communications of the ACM*, **52**:7 (2009), 107–115.

[10] I. V. Maryasov, *The Mixed Axiomatic Semantics Method*, Novosibirsk, Siberian Division of the Russian Academy of Sciences, A. P. Ershov Institute of Informatics Systems, **160**, 2011, http://www.iis.nsk.su/files/preprints/160.pdf.

[11] I. V. Maryasov, V. A. Nepomniaschy, A. V. Promsky, D. A. Kondratyev, "Automatic C Program Verification Based on Mixed Axiomatic Semantics", *Automatic Control and Computer Sciences*, **48**:7 (2014), 407–414, http://link.springer.com/article/10.3103/S0146411614070141.

[12] L. de Moura, N. Bjørner, "Z3: An Efficient SMT Solver", *Tools and Algorithms for the Construction and Analysis of Systems*, 14th International Conference, TACAS 2008 (Budapest, Hungary, March 29–April 6), Lecture Notes in Computer Science, **4963**, Springer, 2008, 337–340, http://link.springer.com/chapter/10.1007/978-3-540-78800-3_24.

[13] V. A. Nepomniaschy, "Symbolic Verification Method for Definite Iteration over Altered Data Structures", *Programming and Computer Software*, **1** (2005), 1–12.

[14] V. A. Nepomniaschy, "Verification of Definite Iteration over Hierarchical Data structures", *Fundamental Approaches to Software Engineering*, Second International Conference, FASE'99 (Amsterdam, The Netherlands, March 22–28), Lecture Notes in Computer Science, **1577**, Springer, 1999, 176–187, http://link.springer.com/chapter/10.1007/978-3-540-49020-3_12.

[15] V. A. Nepomniaschy, "Verification of Definite Iteration over Tuples of Data Structures", *Programming and Computer Software*, **1** (2002), 1–10.

[16] V. A. Nepomniaschy, I. S. Anureev, M. M. Atuchin, I. V. Maryasov, A. A. Petrov, A. V. Promsky, "C Program Verification in SPECTRUM Multilanguage System", *Automatic Control and Computer Sciences*, **45**:7 (2011), 413–420, http://link.springer.com/article/10.3103/S014641161107011X.

[17] V. A. Nepomniaschy, I. S. Anureev, I. N. Mikhaylov, A. V. Promsky, *Towards The Verification of C Programs. Part 3. Translation From C-light To C-light-kernel and Its Formal Proof*, Novosibirsk, Siberian Division of the Russian Academy of Sciences, A. P. Ershov Institute of Informatics Systems, **97**, 2002 (Russian), http://www.iis.nsk.su/files/preprints/097.pdf.

[18] V. A. Nepomniaschy, I. S. Anureev, A. V. Promsky, "Towards Verification of C Programs. C-light Language and Its Formal Semantics", *Programming and Computer Software*, **28** (2002), 314–323.

[19] V. A. Nepomniaschy, I. S. Anureev, A. V. Promsky, "Towards Verification of C Programs. Axiomatic Semantics of The C-kernel Language", *Programming and Computer Software*, **29** (2003), 338–350.

[20] V. A. Nepomniaschy, I. S. Anureev, A. V. Promsky, "Verification-Oriented Language C-Light and Its Structural Operational Semantics", *Perspectives of System Informatics*, 5th International Andrei Ershov Memorial Conference, PSI 2003 (Akademgorodok, Novosibirsk, Russia, July 9–12), Lecture Notes in Computer Science, **2890**, Springer, 2003, 103–111, http://link.springer.com/chapter/10.1007/978-3-540-39866-0_12.

DOI: 10.18255/1818-1015-2015-6-773-782

Элиминация инвариантов циклов для финитной итерации над неизменяемыми структурами данных в Си программах

Марьясов И. В.[1], Непомнящий В. А.

получена 26 октября 2015

Верификация C-программ является актуальной проблемой современного программирования. Для применения известных методов дедуктивной верификации необходимо аннотировать циклы посредством инвариантов, что во многих случаях является трудной задачей. В этой статье мы рассматриваем язык C-light, который является выразительным подмножеством языка C, соответствующего стандарту ISO. Для верификации C-light программ нами были предложены двухуровневый подход [19, 20] и метод смешанной аксиоматической семантики [1, 3, 11]. На первой стадии этого подхода исходная C-light программа транслируется [17] в программу на языке C-kernel [19], который является подмножеством языка C-light. Теорема о корректности этой трансляции была доказана в [10, 11]. По сравнению с C-light в языке C-kernel меньше операторов, что позволяет уменьшить число правил вывода при разработке аксиоматической семантики. На второй стадии этого подхода для программ на языке C-kernel порождаются условия корректности по правилам смешанной аксиоматической семантики [10, 11], которая может содержать несколько правил вывода для одной и той же программной конструкции. В таких случаях правила вывода применяются однозначно в зависимости от контекста. Заметим, что во многих случаях использование смешанной аксиоматической семантики позволяет упростить условия корректности. В этой статье представлено расширение данного подхода, которое включает наш метод верификации для финитной итерации над неизменяемыми структурами данных без выхода из тела цикла в C-light программах. Данный метод содержит новое правило вывода для таких финитных итераций без инвариантов. Это правило было реализовано в генераторе условий корректности. На стадии доказательства используется SMT-решатель Z3 [12]. Рассмотрен пример, иллюстрирующий применение данного подхода.

Статья представляет собой расширенную версию доклада на VI Международном семинаре “Program Semantics, Specification and Verification: Theory and Applications”, Казань, 2015.

Статья публикуется в авторской редакции.

Ключевые слова: Си, инварианты циклов, смешанная аксиоматическая семантика, финитная итерация, неизменяемые структуры данных, спецификация, верификация, логика Хоара

Для цитирования: Марьясов И. В., Непомнящий В. А., "Элиминация инвариантов циклов для финитной итерации над неизменяемыми структурами данных в Си программах", *Моделирование и анализ информационных систем*, **22**:6 (2015), 773–782.

Об авторах:
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, доцент,
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]Эта работа выполнена при поддержке гранта РФФИ 15-01-05974.

Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 783–794
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 783–794

DOI: 10.18255/1818-1015-2015-6-783-794

UDC 519.711

Teaching Formal Models of Concurrency Specification and Analysis

Shilov N. V.[1]

Received October 26, 2015

There is a widespread and rapidly growing interest to the parallel programming nowadays. This interest is based on availability of supercomputers, computer clusters and powerful graphic processors for computational mathematics and simulation. MPI, OpenMP, CUDA and other technologies provide opportunity to write C and FORTRAN code for parallel speed-up of execution without races for resources. Nevertheless concurrency issues (like races) are still very important for parallel systems in general and *distributed systems* in particular. Due to this reason, there is a need of research, study and teaching of formal models of concurrency and methods of distributed system verification.

The paper presents an individual experience with teaching Formal Models of Concurrency as a graduate elective course for students specializing in high-performance computing. First it sketches course background, objectives, lecture plan and topics. Then the paper presents how to formalize (i.e. specify) a reachability puzzle in semantic, syntactic and logic formal models, namely: in Petri nets, in a dialect of Calculus of Communicating Systems (CCS) and in Computation Tree Logic (CTL). This puzzle is a good educational example to present specifics of different formal notations.

The article is published in the author's wording.

Keywords: concurrency and parallelism, formal methods, formal models, Petri nets, calculi for communicating systems, labeled transition systems, reachability problem, temporal logic, model checking

For citation: Shilov N. V., "Teaching Formal Models of Concurrency Specification and Analysis", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 783–794.

On the authors:
Shilov Nikolay Vyacheslavovich, orcid.org/0000-0001-7515-9647, PhD,
A.P. Ershov Institute of Informatics Systems SD RAS,
Lavrent'ev av., 6, Novosibirsk, 630090, Russia,
e-mail: shilov@iis.nsk.su

Acknowledgments:
[1]Reserch is part of Program IV.39.1 "Theory and applied problems of design and implementation of efficient and faultless program systems and information technologies".

Introduction

One of English-to-Russian technical translation problem is about Russian equivalent for English term *concurrency*. Unfortunately the term is translated as *параллелизм* i.e. by the same word as *parallelism*. To make distinction, let us quote a talk by Dan Grossman at Workshop on *Curricula for Concurrency and Parallelism* (Nevada, Oct. 17, 2010) [6]:

> *By parallelism, I mean using extra computational resources to solve a problem faster. By concurrency, I mean correctly and efficiently managing access to shared resources. While using these terms in this way is not entirely standard, the distinction is paramount.*

Rapid growth of parallel computing power raises questions about correctness (i.e. reliability, safety, liveness, etc.) of parallel system and programs. According to the above citation, concurrency is one of the critical issues related to the correctness of parallel systems and programs. It makes important to introduce formal models and methods of concurrency to Computer Science, Software Engineering and Computer Engineering curricula. But a serious problem for curricula development is diversity of individual notations. Another related problem is right choice of introductory, basic and/or advanced teaching/education level.

An elective topic on *Formal Models of Concurrency* at Department of Information Technology of Novosibirsk State University (IT NSU) and at Department of Applied Mathematics of Novosibirsk State Technical University (AM NSTU) was taught in years 2006-2012 during the second (spring) semester of graduate studies. The number of registered students varied from 8 to 16. Typically these students were affiliated with chairs of High-Performance Computing (IT NSU) and Parallel Computation Technologies (AM NSTU). The primary purpose of this course was to introduce basic concepts and means of semantic, syntactic and logic formal models of concurrency that had (already) become classics of Computer Science.

The rest of the paper is as follows. The syllabus of the course is sketched below in the present section; it comprises course background, objectives and topics & lecture plan. The next section 1 introduces a puzzle that is used in the course to illustrate/study/distiguish all formal models covered in the course. In particular, section 2 sketches how to represent the puzzle in Petri nets (semantical formal model), section 3 — in a dialect of Calculus of Communicating Processes (syntax modal), and section 4 — is Computational tree Logic (logic model). Finally we discuss in brief *paradigm of parallel programming* in the concluding section 5. The latest version (for fall semester of academic year 2012-13) of the recommended reading is presented in the References section of the paper and comprises English and Russian papers and books [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19].

0.1. Course Background

At the transition stage to cluster/multiprocessor/multicore architecture and ubiquitous parallelism, the importance (even urgency) of specifying, developing and validating parallel and concurrent behavior is increasing. Formal methods in Computer Science are mathematical theories and techniques for a sound specification, development and verification of soft/hard-ware. The use of formal methods is motivated by the expectation that rigorous mathematical notation and analysis help to understand better functionality and can improve the reliability and robustness. Different formal methods have different types of formal models: semantic models, syntax models, and logical where both syntax and semantics play important roles.

There are numbers of formal models for concurrency, these models have different types. First we have to point out at Petri nets as a purely semantic model. Next we must refer to Communicated Sequential Processes (CSP), an algebraic formal language with fixed syntax and denotational semantics. There exist several calculi that formalize different aspects of concurrency and parallelism: the Calculus of Communicating Systems (CCS), the Pi-Calculus for Communicating and Mobile Systems, the Ambient Calculus and its variations for mobile agents and security, etc. Various types of dynamic, process, and temporal logics are used for specification and verification of parallelism and concurrency.

The course should help Software Engineers and Applied Mathematicians to overview the spectrum of formal models for concurrency and parallelism, get idea of different reasoning techniques that are available for formal verification of concurrent and parallel systems. Students are expected to be familiar with basic concepts of concurrency and parallelism, elements of set theory and propositional Boolean logic.

0.2. Course Objectives

The course intends to help students to achieve the following objectives:

- to understand why we need formal models for concurrency and to know the concept of syntax, semantic and logic formal models;
- to use different formal models of concurrency for modeling concurrent and parallel systems;
- to comprehend definition of Petri nets and to learn reachability and boundedness properties and algorithms for Petri nets;
- to learn syntax and basic semantics of Communicated Sequential Processes (CSP);
- to become acquainted with the Calculus of Communicating Systems (CCS), the Pi-Calculus, and the Ambient Calculus;
- to learn the concept of Labeled Transition Systems (LTS) and to understand why LTS are used for semantics of different formal models;
- to become acquainted with the concept of bisimulation for LTS and to know about Hennessy-Milner logic and its relations to bisimulation;
- to become familiar with syntax and semantics of Computation Tree Logic (CTL) and its use for specification and verification of properties of LTS;
- to discuss ways of introduction of formal models of concurrency and parallelism into Software Engineering practice.

0.3. Topics and Lecture Plan

Introduction. Concepts of syntax, semantic, algebraic and logic formal models (by study of naive set theory, propositional formulas, Boolean algebras, propositional logic). Why do we need formal models for concurrency?

Basics of Petri nets. Definition of marked Petri net. Firing, small/big-step semantics. Reachability problem and reachability tree/graph. Boundedness problem and checking. Problem solving with aid of Petri nets.

Process Algebras. Communicating Sequential Processes (CSP) by A. Hoare. Process algebra of J. Bergstra and J. Klop.

Calculi for concurrency. Calculus of Communicating Systems (CCS) by R. Milner. Pi-Calculus by R. Milner. Ambient Calculus by L. Cardelli. Modeling with aid of calculi.

Labeled Transition Systems. Definition of LTS. Examples of LTS by Petri nets and calculi. Definition of bisimulation and its properties. Bisimulation and Hennessy-Milner Logic.

Computational Tree Logic. CTL syntax and semantics. Using CTL to specify properties of LTS. CTL Model Checking: algorithmic verification of LTS

Conclusion. Do we need a comprehensive model for concurrency and parallelism?

1. One Puzzle for Many Formalisms

The following puzzle *Four Men and a Boat* was used in the course to illustrate several formal models, namely *Petri nets*, a dialect of *Calculus of Communicated Systems* and *Computational Tree Logic.*

> Four men Albert, Conrad, Donald and Edmund are on the left bank of a river and need to move to the right bank by a boat that has 2 seats and one pair of oars. Sporty Albert can cross the river by the boat without a companion in 5 minutes (in one forth or back direction), regular Conrad can do the same in 10 minutes, fatty Donald – in 20 minutes, and fat Edmund – in 25 minutes. When any two men are crossing the river together the pace of the boat is defined by the fattest man in the pair, ex., Albert and Donald together can cross the river in 20 minutes. Question: do these four men can cross the river in one hour?

This is really a very nice puzzle! Typically 8 in 10 students (in my experience) first “prove” that the four men cannot cross the river in one hour. They usually assume that sporty Albert have to accompany (convoy) other men because he is the fastest and it would be better him to transport the boat back every time; under this assumption transportation of 4 men takes 1 hour and 5 minutes.

The author also made this assumption when heard the puzzle for the first time 17 years ago, and was very much confused when Andrei Sabelfeld (http://www.cse.

chalmers.se/~andrei/, he was a student at the time of the story) told him that it is wrong. In turn Andrey simply gave the following scenario how men can cross the river in one hour:

- first Albert and Conrad cross the river together in 10 minutes, then Albert transports the boat back in 5 minutes;
- next Donald and Edmund cross the river together in 25 minutes and Conrad transports the boat back in 10 minute;
- finally Albert and Conrad cross the river together again in 10 minutes.

So this puzzle gave a good lesson to author to be skeptical about "obvious" assumptions.

2. Puzzle in Petri nets

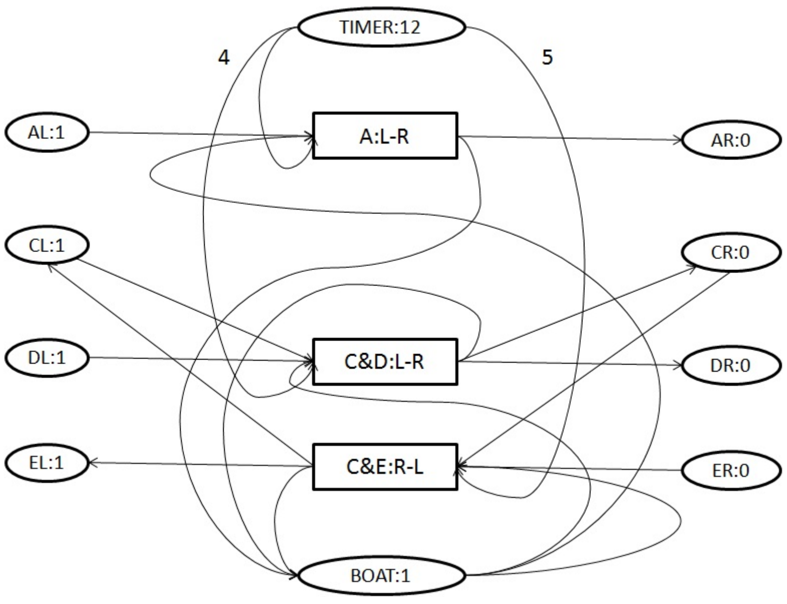


Fig 1. Fragment of net model for Four Men and a Boat Puzzle

A sketch of a marked Petri net that models the puzzle is presented on the figure 1. One can see on this figure

- a place TIMER with initial marking 12 tokens each of which models 5 minutes (because all time values in the puzzle are dividable by 5);
- a place BOAT with initial marking 1 that models the boat;
- places AL, CL, DL and EL with initial marking 1 each; the initial marking models that initially Albert (A), Conrad (C), Donald (D) and Edmund (E) are on the left (L) bank of the river;

- places AR, CR, DR and ER with initial markings 0 each; the initial marking models that initially Albert (A), Conrad (C), Donald (D) and Edmund (E) are absent on the right (R) bank of the river;

- three sample transitions that model situations when Albert moves from left bank to the right (A:L-R), when Bob and Conrad move together from left bank to the right (C&D:L-R), and when Bob and Donald move together from right bank to the left (C&E:R-L) by boat; omitted 17 transitions are similar;

- several edges without tags and two edges tagged by integers 4 and 5: edges without tags have multiplicity 1, edges tagged by integers have multiplicity according to their tags; the multiplicity represents the number of 5 minutes intervals that is required for the corresponding transition;

- firing of each transition models a move from one bank to another by a man or by a pair of men in the boat.

The presented marked net is bounded, it can be checked by construction of Karp-Miller coverage tree. The background (unmarked) net is structurally bounded (i.e. it is bounded for every initial marking), the structural boundedness of the net is obvious[1].

In terms of Petri nets, to solve the puzzle means that the following final marking is reachable from the initial one (TIMER:0, AL:0, CL:0, DL:0, EL:0, AR:1, CR:1, DR:1, ER:1, BOAT:1). Since the net model is bounded, the set of all reachable markings is finite[2]; so the puzzle can be solved by construction of the reachability graph for the marked net. Let us observe that this reachability graph is an example of labeled transition system where nodes are reachable markings of the net and edges are transition firings.

3. Puzzle in CCS

Let us describe a simplified dialect of Calculus of Communicating Systems. Syntax of the dialect is defined in terms of the following three context-free grammars:

$$\begin{aligned} S &::= \langle empty\rangle \mid N = P; S \\ P &::= 0 \mid A.P \mid (P+P) \mid (P|P) \mid N \\ A &::= R \mid I \mid O \end{aligned}$$

where

- N, R, I and O are disjoint finite alphabets which elements are called *process names*, *regular*, *input* and *output* (*action*) *symbols*,

- words generated from symbol P are called *processes*, and words generated from from symbol S are sets of *definitions* (separated by semicolon).

[1]Sorry for claiming that something is “obvious”. This time the claim follows from the net structure: for every marking every firing of a transition reduces the total number of tokens.

[2]It contains not more than 208 reachable states.

Process constructors ".", "+" and "|" are conventionally called *sequential*, *nondeterministic* and *parallel* compositions. A set of definitions is consistent if every name that occurs in the system has single definition in the set.

Input and output alphabets are mutually complementary: they are equipped by a bijection function $(\ )^c$ such that

- for every input symbol i the complement i^c is an output symbol and $(i^c)^c = i$,
- for every output symbol o the complement o^c is an input symbol and $(o^c)^c = o$.

Let us specify the puzzle in terms of our dialect of CCS. Let

$$TIMER = \underbrace{tick.tick.tick.tick.tick.tick.tick.tick.tick.tick.tick.tick.}_{12\ times}0$$

be definition of a process name $TIMER$ where *tick* is an output symbol (to represent time interval of 5 minutes) with the complementary input symbol *tack*. This input symbol is used in process definitions affiliated with men in the puzzle. For example, in definitions for two process names AL and AR (that correspond to Albert on the left and on the right banks) follow below:

$$AL = (acqBL.tack.AR\ +\ ALackCL.AR\ +\ ALackDL.AR\ +\ ALackEL.AR)$$
$$AR = (done.0\ +\ acqBR.tack.AL\ +$$
$$+\ ARackCR.AL\ +\ ARackDR.AL\ +\ ARackER.AL)$$

where

- *done* is a regular action symbol;
- $acqBL$ and $acqBR$ are output symbols (to represent that a man acquires the boat located on the left/right back respectively) with complementary input symbols $relBL$ and $relBR$ (to represent boat release);
- $ALackCL$, $ALackDL$, $ALackEL$ and $ARackCR$, $ARackDR$, $ARackER$ are input symbols (to represent that Albert acknowledges an invitation of Conrad, Donald or Edmund with the same location to cross the river together) with complimentary symbols $CLaskAL$, $DLaskAL$, $ELaskAL$ and $CRaskAR$, $DRaskAR$, $ERaskAR$ respectively (to represent invitations).

The intended semantics of these two definitions is behavior of Albert on the left and on the right banks.

- AL specifies that on the left bank he has the following 4 disjoint options:
  - acquire the boat ($acqBL$), cross the river (synchronizing *tack* with *tick*) and then proceed further on the right back according to AR;
  - acknowledge invitation to use the boat acquired by Conrad ($ALackCL$) to cross the river and then proceed further on the right back according to AR;
  - acknowledge invitation to use the boat acquired by Donald ($ALackDL$) to cross the river and then proceed further on the right back according to AR;

  - acknowledge invitation to use the boat acquired by Edmund ($ALackEL$) to cross the river and then proceed further on the right back according to AR.

- AR specifies that on the right bank he has the following 5 disjoint options:
  - stop any further activities and remain on the right bank (*done*);
  - acquire the boat ($acqBR$), cross the river (synchronizing *tack* with *tick*) and then proceed further on the left back according to AL;
  - acknowledge invitation to use the boat acquired by Conrad ($ARackCR$) to cross the river and then proceed further on the left back according to AL;
  - acknowledge invitation to use the boat acquired by Donald ($ARackDR$) to cross the river and then proceed further on the left back according to AL;
  - acknowledge invitation to use the boat acquired by Edmund ($ARackER$) to cross the river and then proceed further on the left back according to AL.

Definitions for names CL and CR (that correspond to Conrad on the left and on the right banks), DL and DR (that correspond to Donald on the left and on the right banks), and EL and ER (that correspond to Edmund on the left and on the right banks) are presented in the table 1.

Table 1. Definitions that model Conrad, Donald and Edmund behavior

$$\begin{aligned}
CL &= acqBL.tack.tack.(CR + CLaskAL.CR) + CLackDL.CR + CLackEL.CR\\
CR &= done.0 + acqBR.tack.tack.(CL + CRackAR.CL) + CRackDR.CL + CRackER.CL\\
DL &= acqBL.tack.tack.tack.tack.(DR + DLaskAL.DR + DLaskCL.DR) + CLackEL.CR\\
DR &= done.0 + acqBR.tack.tack.tack.tack(DL + DRaskAR.DLDRaskCR.DL)+\\
&\qquad +DRackER.DL\\
EL &= acqBL.tack.tack.tack.tack.tack.\\
&\qquad (ER + ELaskAL.ER + ELaskCL.ER + ELaskDL.ER)\\
ER &= done.0 + acqBR.tack.tack.tack.tack.tack.\\
&\qquad (EL + ERaskAR.EL + ERaskCR.EL + ERaskDR.EL)
\end{aligned}$$

The last two definitions for names BL and BR correspond to the boat on the left and on the right banks:

$$BL = relBL.BR \text{ and } BR = (done.0 + relBR.BL).$$

The initial configuration of the puzzle can be represented as the following process

$$(TIMER|AL|CL|DL|EL|BL).$$

Reduction rules are very standard for CCS:

- if r is a regular action symbol then $r.\alpha \to \alpha$,
- if $\alpha \to \gamma$ then $(\alpha + \beta) \to \gamma$ and $(\alpha|\beta) \to (\gamma|\beta)$,
- if s is an input/output symbol then $(s.\alpha|s^c.\beta|\gamma) \to (\alpha|\beta|\gamma)$,

as well as the following congruencies:

- $\alpha + \beta \equiv \beta + \alpha$ and $\alpha + (\beta + \gamma) \equiv (\alpha + \beta) + \gamma$,
- $\alpha|0 \equiv \alpha$, $\alpha|\beta \equiv \beta|\alpha$, and $\alpha|(\beta|\gamma) \equiv (\alpha|\beta)|\gamma$,
- if S is a set of name definitions and $n = \alpha$ is one of them then[3] $n \overset{S}{\equiv} \alpha$.

In terms of the presented CCS-dialect, to solve the puzzle means that the final process 0 is reachable (assuming system of name definitions that comprises all definitions listed above are provided) in the reduction graph for the initial process $(TIMER \mid AL \mid CL| DL \mid EL \mid BL)$. This graph is finite because every nondeterministic branch with recursion in any of our definitions contains at least one *tack* that must be synchronize with *tick*, but the total amount of ticks is 12 (according to definition of $TIMER$). So the puzzle can be solved in terms of CCS by construction of the reduction graph. Let us observe that this graph is also an example of labeled transition system where nodes are process configurations (that are reducts of the initial process) and edges are reductions.

4. Puzzle in CTL

As follows from sections 2 and 3, the puzzle can be reduced to the reachability problem for finite graphs (label transition systems in particular) with aid of Petri nets (semantic model) or CCS (syntactic model). Moreover, the puzzle can be solved on the fly at time of constructing the corresponding labeled transition system.

But let us recall that students who try to solve the puzzle usually believe that Albert have to accompany other men for accelerating transportation. In other words they make the following *belief assumption*:

If a positive solution exists,
then there exists a solution where Albert convoys other men
until all are on the right bank.

There are two ways how to refute this belief: human-oriented or computer-aided.

- Human-oriented way in this case comprises two steps: first someone should solve the puzzle (it was Andrey Sabelfeld in my case), then prove manually impossibility of a solution where Albert convoys other men (that is very easy).
- Computer-aided approach needs to build the corresponding labeled transition system first, then to modify a reachability algorithm to check whether there exists a path where Albert is always on the move.

The first step of a computer-aided approach can be carried out in many ways: for example one may construct reachability graph for the Petri net (that model the puzzle), or reduction graph for the corresponding CCS specification. The next step of the approach may be generalized: it makes sense to build an efficiently decidable graph *query language* so that the standard reachability (and many its modifications) becomes just a special query of this language.

[3]Supscript S may be omitted when the system is implicit.

There are many query languages of this sort indeed, Computation Tree Logic (CTL), Linear Temporal Logic, μ-Calculus for instance. For example, in CTL the belief assumption can be represented by CTL formula presented in the table 2. Here $Albert\_at\_Left$, $Conrad\_at\_Left$, ..., $Edmund\_at\_Right$ and $Albert\_on\_Move$ are propositional variables with natural interpretation in the labeled transition system.

Table 2. CTL specification for the *belief assumption* about the puzzle

$$\begin{aligned}&(Albert\_at\_Left \,\&\, Conrad\_at\_Left \,\&\\ &\&\, Donald\_at\_Left \,\&\, Edmund\_at\_Left \,\&\\ &\&\, Boat\_at\_Left \,\&\, Timer\_is\_Set) \rightarrow\\ &\qquad\rightarrow (\mathbf{EF}(Albert\_at\_Right \,\&Conrad\_at\_Right \,\&\\ &\qquad\&\, Donald\_at\_Right \,\&\, Edmund\_at\_Right) \rightarrow\\ &\qquad\rightarrow \mathbf{E}(Albert\_on\_Move \;\mathbf{U}\; (Albert\_at\_Right \,\&\, Conrad\_at\_Right \,\&\\ &\qquad\qquad\&\, Donald\_at\_Right \,\&\, Edmund\_at\_Right)))\end{aligned}$$

5. Parallel Programming Paradigm

Paradigm is an approach to problem formulation/formalization and the ways to solve them. The term comes from Greek and means *pattern*, *example* (nouns), *exhibit*, *represent* (verbs). A contemporary meaning of the term is due to well-known book by Tomas Kuhn [11]. Robert Floyd was the first who had explicitly used the term in the Computer Science context. In particular, he addressed *Paradigms of Programming* in his Turing Award Lecture in 1968 [5]. Unfortunately, R. Floyd had not defined explicitly this concept. Peter Van Roy has published in 2009 a taxonomy *The principal programming paradigms* (at `http://www.info.ucl.ac.be/~pvr/paradigms.html`) with 27 different paradigms. He also suggested the following definition [16]:

> *A programming paradigm is an approach to programming a computer based on a mathematical theory or a coherent set of principles. Each paradigm supports a set of concepts that makes it the best for a certain kind of problem.*

The definition suggested by Peter Van Roy has been refined in [18] as follows.

- Programming paradigms are alternative approaches (patterns) to formalization of information problems (problem statements), data presentation, handling, and processing.
- They are fixed in a form of formal (mathematical) theory and accumulated in programming languages.
- Programming value of a paradigm may be characterized by a set of programming problems/application areas that the paradigm fits better than the other ones.
- Educational value of programming paradigms is to teach to think different about programming problems and to select the best paradigm to solve any particular problem.

This definition assumes that teaching/learning formal models/theories is obligatory for mastering/comprehending any programming paradigm, the parallel programming in particular. And the puzzle about four men and a boat helps to teach and learn some formal models and theories of concurrency.

References

[1] *Handbook of Process Algebra*, eds. J. A. Bergstra, A. Ponse, S. A. Smolka, Elsevier, Amsterdam, 2001.

[2] L. Cardelli, A. D. Gordon, “Mobile ambient”, Lecture Notes in Computer Science, **1378**, Springer-Verlag, Berlin, Heidelberg, 1998, 140–155.

[3] L. Cardelli, “Mobility and Security.”, *Foundations of Secure Computation*, Proceedings of the NATO Advanced Study Institute on Foundations of Secure Computation, IOS Press, Amsterdam, 2000, 3–37.

[4] E. M. Jr. Clarke, O. Grumberg, D. A. Peled, *Model Checking*, MIT Press, Cambridge, Massachusetts, 1999.

[5] R. W. Floyd, “The paradigms of Programming”, *Communications of ACM*, **22** (1979), 455–460.

[6] D. Grossman, “Ready-For-Use: 3 Weeks of Parallelism and Concurrency in a Required Second-Year Data-Structures Course”, SPLASH 2010 Workshop on Curricula for Concurrency and Parallelism (Reno, Nevada, USA, Oct. 17, 2010), https://homes.cs.washington.edu/~djg/papers/grossmanSPAC_position2010.pdf.

[7] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, Upper Saddle River, New Jersey, 1985, (This book was updated by Jim Davies at the Oxford University Computing Laboratory in 2004 and the new edition is available at the http://www.usingcsp.com/).

[8] Yu. G. Karpov, *Model Checking. Verificaciya parallelnyh i raspredelennyh programmnyh system*, BHV-Peterburg, Saint Petersburg, 2010, (In Russian).

[9] V. E. Kotov, *Seti Petri*, Nauka, Moscow, Russia, 1987, (In Russian).

[10] Z. Manna, A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag,, New York, 1992.

[11] T. S. Kuhn, *The structure of Scientific Revolutions*, Univ. of Chicago Press, Chicago and London, 1996, (3rd Edition).

[12] R. Milner, *Communicating and Mobile Systems: the Pi-Calculus*, Cambridge University Press, Cambridge, England, 1999.

[13] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, **92**, Springer-Verlag, Berlin, Heidelberg, 1980.

[14] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Upper Saddle River, New Jersey, 1981.

[15] W. Reisig, *A Primer in Petri Net Design*, Springer-Verlag, Berlin, Heidelberg, 1992.

[16] P. van Roy, “Programming Paradigms for Dummies: What Every Programmer Should Know”, *New Computational Paradigms for Computer Music,*, eds. G. Assayag, A. Gerzso, IRCAM/Delatour, France, 2009, 9–38.

[17] N. V. Shilov, K. Yi, “How to Find a Coin: Propositional Program Logics Made Easy”, *Current Trends in Theoretical Computer Science.* V. 2, World Scientific, Singapore, 2004, 181–214.

[18] N. V. Shilov et al., “Development of the Computer Language Classification Knowledge Portal”, *Ershov Informatics Conference PSI’11*, Lecture Notes in Computer Science, **7162**, Springer-Verlag, Berlin, Heidelberg, 2012, 340–348.

[19] C. Stirling, *Modal and Temporal Properties of Processes*, Springer-Verlag, New York, 2001.

DOI: 10.18255/1818-1015-2015-6-783-794

О преподавании формальных моделей и алгоритмов анализа параллельных систем

Шилов Н. В.[1]

получена 26 октября 2015

В настоящее время наблюдается огромный практический интерес к параллельному программированию. Этот интерес обусловлен доступностью супер-ЭВМ, компьютерных кластеров и мощных графических процессоров для массового использования в вычислительной математике и компьютерном моделировании. Кроме того, такие технологии параллельного программирования, как MPI, OpenMP и CUDA, позволяют использовать безопасным образом опыт программирования на классических языках Си и FORTRAN для ускорения вычислений, избегая конфликтов (“гонок”) из-за ресурсов. Однако такой прогресс параллельного программирования не означает, что конкуренция из-за ресурсов не может возникать в параллельных общего вида, в так называемых *распределенных системах* в частности. Поэтому остается актуальным изучение и преподавание формальных моделей параллелизма и средств верификации поведенческих свойств параллельных (распределенных) систем.

В статье представлен опыт преподавания специального курса по формальным моделям параллелизма для магистрантов и аспирантов, специализирующихся в области высокопроизводительных вычислений. Сначала в статье дан обзор курса в целом, предварительных знаний, необходимых для этого курса, целей и задач курса, представлен план лекций и список рекомендованной литературы. Затем представлен пример одной поучительной головоломки (на достижимость в пространстве состояний) и ее формализации средствами семантических, синтаксических и логических моделей, как-то: сетями Петри, средствами исчисления параллельных взаимодействующих процессов (CCS) и темпоральной логики CTL. Эта головоломка — хороший пример для того, чтобы показать специфику и пользу каждого из рассмотренных формализмов.

Статья представляет собой расширенную версию доклада на VI Международном семинаре “Program Semantics, Specification and Verification: Theory and Applications”, Казань, 2015.

Статья публикуется в авторской редакции.

Ключевые слова: системы с параллелизмом, формальные методы, формальные модели, сети Петри, исчисления взаимодействующих процессов, помеченные системы переходов, проблема достижимости, темпоральная логика, верификация моделей

Для цитирования: Шилов Н. В., "О преподавании формальных моделей и алгоритмов анализа параллельных систем", *Моделирование и анализ информационных систем*, **22**:6 (2015), 783–794.

Об авторах:
Шилов Николай Вячеславович, orcid.org/0000-0001-7515-9647,
канд. физ.-мат. наук, старший научный сотрудник,
Институт систем информатики им. А. П. Ершова СО РАН
630090 Россия, г. Новосибирск, пр. Лаврентьева, 6,
e-mail: shilov@iis.nsk.su

Благодарности:
[1]Работа выполнена в рамках программы IV.39.1 «Теоретические и прикладные проблемы создания эффективных надежных программных систем и информационных технологий».

Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 795–817
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 795–817

DOI: 10.18255/1818-1015-2015-6-795-817

UDC 51

Formal Diagonalisation of Lax-Darboux Schemes

A.V. Mikhailov

Received November 28, 2015

We discuss the concept of Lax-Darboux scheme and illustrate it on well known examples associated with the Nonlinear Schrödinger (NLS) equation. We explore the Darboux links of the NLS hierarchy with the hierarchy of Heisenberg model, principal chiral field model as well as with differential-difference integrable systems (including the Toda lattice and differential-difference Heisenberg chain) and integrable partial difference systems. We show that there exists a transformation which formally diagonalises all elements of the Lax-Darboux scheme simultaneously. It provides us with generating functions of local conservation laws for all integrable systems obtained. We discuss the relations between conservation laws for systems belonging to the Lax-Darboux scheme.

Keywords: formal diagonalisation, Lax-Darboux schemes, nonlinear Schrödinger equation, NLS

For citation: Mikhailov A. V., "Formal Diagonalisation of Lax-Darboux Schemes", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 795–817.

On the authors:
Alexander V Mikhailov – Professor, University of Leeds, School of Mathematics (Leeds, UK)
University of Leeds, Leeds, LS2 9JT, UK
e-mail: A.V.Mikhailov @ leeds.ac.uk

1. Introduction

Often, integrable partial differential, differential-difference and partial difference equations can be regarded as parts of an algebraic structure which we call a Lax-Darboux scheme. In this context the Lax representations for the integrable partial differential equation (PDE) and a hierarchy of its symmetries form a Lax structure. Darboux transformations for the corresponding Lax operators are automorphisms of this Lax structure resulting in a chain of Bäcklund transformations for the PDE and its symmetries. The latter represents an integrable system of differential-difference equations (DΔEs) and its symmetries (often non-evolutionary). The Bianchi permutability conditions for Darboux transformations represent a system of partial difference equations (PΔEs). This system possesses an infinite hierarchy of commuting symmetries (the above mentioned DΔEs) and thus it is an integrable partial difference system in its own right. There are many journal publications and monographs focusing on certain aspects of this big picture [1, 2, 3, 4, 5]. In particular, paper [4] contains a good collection of Lax-Darboux representations recursion operators for integrable differential-difference equations. In this paper we discuss the ways in which PDEs, DΔEs and PΔEs belonging to the same Lax-Darboux scheme share the same hierarchy of local conservation laws.

The main idea standing behind our theory is a formal diagonalisation of the Lax-Darboux scheme. We show that there exists a formal (i.e. in the form of a formal series) gauge transformation which simultaneously diagonalises (or brings to a block-diagonal form) the Lax operators of the Lax structure and Darboux matrices associated with Darboux transformations. It provides us with a regular method for recursive derivation of a hierarchy of local conservation laws for the nonlinear differential and difference systems associated with the Lax-Darboux scheme.

The method of formal diagonalisation of differential operators can be found in the classical literature concerning asymptotic expansion [6]. In application to Lax representations for partial differential equations and recursive derivation of the hierarchy of local conservation laws, there is a neat and instructive exposition of the method [7]. It has been successfully used in the Symmetry Approach to classify of integrable partial differential equations [8]. Here we extend the method to Darboux transformations and in this way to differential-difference and partial difference integrable systems. We shall explain the method using the Lax-Darboux scheme associated with the Nonlinear Schrödinger equation. Its generalisations to other Lax-Darboux schemes is rather straightforward. This paper is based on a lecture courses given by the author in the Bashkir State University (Ufa, 2012) and as a part of MAGIC course on Integrable systems (UK, 2014), a number of conference talks (Ufa, October 2012; Moscow, November 2012 [9]; Cambridge, July 2013 [10]) where the concept of Lax-Darbiux scheme and formal diagonalisation approach were originally presented. This method has proven to be useful in a many applications(see for example [5, 11, 12]).

2. Lax-Darboux scheme for the Nonlinear Schrödinger equation

In this paper we consider Lax integrable equations, i.e. equations which can be integrated using the Inverse Spectral Transform method. For such equations we can build up a Lax-Darboux scheme with the following objects:

- the *Lax structure*, which is an infinite sequence of Lax operators whose commutativity conditions are equivalent to the equation and a hierarchy of commuting symmetries;
- *Darboux transformations*, which are automorphisms of the Lax structure;
- *Bäcklund transformations*, which are follow from the compatibility conditions of the Lax structure and Darboux transformation. They can be regarded as *integrable differential-difference systems*;
- the conditions of Bianchi permutability for the Darboux transformations, which lead to systems of *integrable partial difference equations*;
- Ajacent Lax structures associated with a Darboux transformation which lead to adjacent symmetries of these differential-difference and partial difference equations and are integrable differential-difference equations in their own right.

In this section we would like to give explicit representations of all listed above objects in the case of the Nonlinear Schrödinger (NLS) equation.

2.1. Lax structure of the Nonlinear Schrödinger Equation

The Nonlinear Schrödinger Equation is a system of two partial differential equations

$$2p_t = p_{xx} - 8p^2q, \qquad 2q_t = -q_{xx} + 8q^2p \tag{1}$$

where x, t are independent variables. In the literature the term Nonlinear Schrödinger Equation usually stands for one complex equation of the form

$$iq_t = q_{xx} \pm 2|q|^2 q,$$

which can be obtained from (1) after a change of variables $t \to 2it$, $x \to 2ix$ and reduction $p = \mp q^*$. In this paper we shall use equation (1) to illustrate the method, since the reduction condition would add some inessential technicalities.

It has been shown by Zakharov and Shabat [13] that the system of equations (1) is equivalent to the commutativity condition $[L(p, q), A(p, q)] = 0$ for two linear differential operators

$$L(p, q) = D_x - U, \quad A(p, q) = D_t - V, \tag{2}$$

where D_x and D_t are operators of differentiation in x and t respectively,

$$U = \lambda J + \begin{pmatrix} 0 & 2p \\ 2q & 0 \end{pmatrix}, \; J = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \; V = \lambda U - \begin{pmatrix} 2pq & -p_x \\ q_x & -2pq \end{pmatrix}, \tag{3}$$

and λ is a spectral parameter. Linear operators $L(p,q)$, $A(p,q)$ form a Lax pair (or a Lax representation) for equation (1).

The NLS system (1) admits an infinite hierarchy of commuting symmetries

$$p_{t_k} = f^k, \quad q_{t_k} = g^k, \qquad k = 0, 1, 2, \dots \tag{4}$$

where

$$\begin{array}{ll} f^0 = 2p, & g^0 = -2q, \\ f^1 = p_x, & g^1 = q_x, \\ f^2 = \dfrac{1}{2}p_{xx} - 4p^2q, & g^2 = -\dfrac{1}{2}q_{xx} + 4q^2p \qquad \text{NLS}, \\ f^3 = \dfrac{1}{4}p_{xxx} - 6pqp_x, & g^3 = \dfrac{1}{4}q_{xx} - 6qpq_x, \end{array}$$

$$f^4 = \frac{1}{8}p_{xxxx} - 4pqp_{xx} - 3qp_x^2 - 2pp_xq_x - p^2q_{xx} + 12p^3q^2,$$
$$g^4 = \frac{1}{8}q_{xxxx} - 4qpq_{xx} - 3pq_x^2 - 2qq_xp_x - q^2p_{xx} + 12q^3p^2, \ \dots$$

All functions $f^k, g^k \in [\mathbb{C}, p, q; D_x]$ are differential polynomials over the field $\mathbb{C}$ of variables $p^{(0)} = p,\ q^{(0)} = q$ and their x-derivatives $p^{(1)} = p_x,\ q^{(1)} = q_x,\ p^{(2)} = p_{xx},\ q^{(2)} = q_{xx},\ \dots$ and

$$D_x = \sum_{n=0}^{\infty} p^{(n+1)} \frac{\partial}{\partial p^{(n)}} + q^{(n+1)} \frac{\partial}{\partial q^{(n)}}.$$

By (generalised) symmetries of the NLS equation (1) we understand derivations

$$D_{t_k} = \sum_{n=0}^{\infty} D_x^n(f^k) \frac{\partial}{\partial p^{(n)}} + D_x^n(g^k) \frac{\partial}{\partial q^{(n)}}$$

commuting with $D_t = D_{t_2}$ and D_x

$$[D_{t_2}, D_{t_k}] = 0, \qquad [D_x, D_{t_k}] = 0.$$

Symmetries are called commuting if the corresponding derivations commute. It is sufficient to verify that $D_{t_n}(f_m) = D_{t_m}(f_n)$ and $D_{t_n}(g_m) = D_{t_m}(g_n)$. Motivations and general definition of symmetries one can find in [14, 15].

Each symmetry from this hierarchy has a Lax representation

$$p_{t_k} = f^k, \quad q_{t_k} = g^k \ \Leftrightarrow\ [L(p,q), A^k(p,q)] = 0 \tag{5}$$

with the same operator $L(p,q) = D_x - U$ and $A^k(p,q) = D_{t_k} - V^k$ where matrices V^k can be found recursively starting from $V^0 = J$ and for $k \geq 1$

$$V^{k+1} = \lambda V^k - \frac{1}{2}D_x(V^k)J - \frac{1}{2}[V^k, U]J - \frac{1}{2}D_x^{-1}\mathrm{Tr}(UD_x(V^k))J. \tag{6}$$

Here D_x^{-1} stands for integration in x. It can be rigorously proven that $\mathrm{Tr}(UD_x(V^k)) \in \mathrm{Im}\, D_x$ for any k, thus the integral can be evaluated and the result belongs to the differential ring $[\mathbb{C}; p, q; D_x]$. The constants of integration can be chosen arbitrary, or fixed by the condition $V^k|_{p(x)=q(x)=0} = \lambda^k J$. In the latter case

$$V^0 = J,\ V^1 = U,\ V^2 = V,\ V^3 = \lambda V^2 + \frac{1}{2}\begin{pmatrix} 2pq_x - 2qp_x & p_{xx} - 8p^2q \\ q_{xx} - 8q^2p & 2qp_x - 2pq_x \end{pmatrix}, \dots .$$

Recursion relation (6) can be simplified and represented in the form

$$V^0 = J, \qquad V^{n+1} = \lambda V^n + B_n, \quad n = 0, 1, \dots$$

where the λ–independent matrices B_n can be found recursively

$$B_0 = \begin{pmatrix} 0 & 2p \\ 2q & 0 \end{pmatrix}, \quad B_{k+1} = \frac{J}{2}\left(D_x(B_k) + [B_k, B_0] - D_x^{-1}\mathrm{Tr}(B_0 D_x(B_k))\right).$$

The set of Lax operators $\{L(p,q), A^k(p,q),\ k \in \mathbb{Z}_{\geq 0}\}$ and corresponding compatible partial differential equations (commuting symmetries) $\{(f^k, g^k),\ k \in \mathbb{Z}_{\geq 0}\}$ form the Lax structure for the Nonlinear Schrödinger equation (1).

2.2. Darboux and Bäcklund transformations for NLS

Since all linear differential operators $\{L(p,q), A^k(p,q),\ k \in \mathbb{Z}_{\geq 0}\}$ commute with each other, there exists a common fundamental solution Ψ of the linear problems

$$L(p,q)\Psi = 0, \qquad A^k(p,q)\Psi = 0. \tag{7}$$

We shall study a transformation $\mathcal{S}$ of a fundamental solution

$$\mathcal{S}: \ \Psi \mapsto \overline{\Psi} = M\Psi, \qquad \det M \not\equiv 0 \tag{8}$$

such that the matrix function $\overline{\Psi}$ is a fundamental solution of the linear problems

$$L(\bar{p},\bar{q})\bar{\Psi} = 0, \qquad A^k(\bar{p},\bar{q})\bar{\Psi} = 0. \tag{9}$$

with new “updated potentials” $\bar{p}, \bar{q}$. In the literature this type of transformation is often referred to as a Darboux transformation and the matrix M is called the Darboux matrix. We shall assume that a Darboux matrix M is a rational function of the spectral parameter λ, whose entry may depend on $p, q, \bar{p}, \bar{q}$ and may also depend on some auxiliary function(s) h or constant parameter(s) (examples will be given later in this section). The given description of a Darboux transformation can be cast into a rigorous definition using elements of differential-difference ring theory. We wish to avoid the introduction of these concepts at the present time to make the paper accessible to a wider community.

It follows from (7), (8), (9) that transformation $\mathcal{S}$ can be extended to the set of Lax operators:

$$\begin{aligned} &\mathcal{S}: L(p,q) \mapsto L(\bar{p},\bar{q}) = ML(p,q)M^{-1}, \\ &\mathcal{S}: A^k(p,q) \mapsto A^k(\bar{p},\bar{q}) = MA^k(p,q)M^{-1}, \quad k = 0, 1, \dots\ . \end{aligned} \tag{10}$$

Let us show it for the first equation. Indeed, we have

$$D_x\bar{\Psi} = \mathcal{S}(U)\bar{\Psi} \Rightarrow D_x(M\Psi) = \mathcal{S}(U)M\Psi \Rightarrow D_x(M) - \mathcal{S}(U)M + MU = 0$$

and thus $L(\bar{p},\bar{q})M - ML(p,q) = 0$. Here we use notation $\mathcal{S}(U)$ for matrix U (3) in which variables p, q are replaced by $\bar{p}, \bar{q}$. Equations (10) are equivalent to a compatible system of equations for M

$$D_x(M) = \mathcal{S}(U)M - MU, \tag{11}$$

$$D_{t_k}(M) = \mathcal{S}(V^k)M - MV^k.. \tag{12}$$

It follows from (10) that a Darboux transformation $\mathcal{S}$ can be regarded as an automorphism of the Lax structure. It maps the set of commuting Lax operators into another commuting set

$$\mathcal{S} : \{L(p,q), A^k(p,q);\ k\in\mathbb{Z}_{\geq 0}\} \mapsto \{L(\bar p,\bar q),\ A^k(\bar p,\bar q);\ k\in\mathbb{Z}_{\geq 0}\},$$

and results in a Bäcklund transformation which transforms a solution p,q of the NLS hierarchy into a new solution

$$\mathcal{S} : (p,q) \mapsto (\bar p,\bar q).$$

Equations (11), (12) follow from the conditions that the map $\mathcal{S}$ and derivations D_x, D_{t_k} commute

$$\mathcal{S}L = MLM^{-1} := \mathrm{Ad}_M L,\quad \mathcal{S}A^k = MA^kM^{-1} := \mathrm{Ad}_M A^k,\qquad k=0,1,\dots\ .$$

Darboux transformations are obviously invertible and a composition of two Darboux transformations $\mathcal{S}_1,\mathcal{S}_2$ with Darboux matrices M_1,M_2

$$\mathcal{S}_2\circ\mathcal{S}_1 : \Psi \mapsto \mathcal{S}_1(M_2)M_1\Psi$$

is a Darboux transformation. There is a problem to describe all *elementary* Darboux transformations for a given Lax structure, such that any rational Darboux transformation can be represented as a composition of the elementary ones. In the cases of the Lax structure corresponding to the Korteweg-de Vries equation and the Nonlinear Schrödinger equation the solution of this problem can be found in [16]. In particular, it has been shown that any Darboux transformation of the Lax operators L, A (2) can be represented as a composition of elementary Darboux transformations $\mathcal{J}_\beta, \mathcal{S}_\alpha, \mathcal{T}_h$ with matrices

$$\mathcal{J}_\beta : J_\beta = \mathrm{diag}(\beta,\beta^{-1}), \tag{13}$$

$$\mathcal{S}_\alpha : M_\alpha = \begin{pmatrix} \lambda+p\mathcal{S}_\alpha(q)-\alpha & p\\ \mathcal{S}_\alpha(q) & 1\end{pmatrix}, \tag{14}$$

$$\mathcal{T}_h : N_h = \begin{pmatrix} \lambda+h & p\\ \mathcal{T}_h(q) & 0\end{pmatrix}, \tag{15}$$

and their inverse transformations with a certain choice of constant complex parameters β,α.

It follows from (11) that

$$\mathcal{J}_\beta L = \mathrm{Ad}_{J_\beta} L \ \Leftrightarrow\ \mathcal{J}_\beta(p) = \beta^2 p,\ \mathcal{J}_\beta(q) = \beta^{-2}q; \tag{16}$$

$$\mathcal{S}_\alpha L = \mathrm{Ad}_{M_\alpha} L \ \Leftrightarrow\ \begin{cases} p_x = 2\mathcal{S}_\alpha(p) - 2p^2\mathcal{S}_\alpha(q) + 2\alpha p,\\[2ex] \mathcal{S}_\alpha(q_x) = -2q + 2p\mathcal{S}_\alpha(q^2) - 2\alpha\mathcal{S}_\alpha(q);\end{cases} \tag{17}$$

$$\mathcal{T}_h L = \mathrm{Ad}_{N_h} L \ \Leftrightarrow\ \begin{cases} p_x = -2hp,\ h_x = 2(\mathcal{T}_h - 1)(pq),\\[2ex] p\mathcal{T}_h(q) = 1.\end{cases} \tag{18}$$

The first map (16)

$$\mathcal{T}_\beta : (p,q) \mapsto (\beta^2 p, \beta^{-2} q)$$

is nothing but a point symmetry of the NLS equation (1).

Equation (17) is a Bäcklund transformation (the x part of the Bäcklund transformation) for the NLS equation (1). Starting from a solution p, q of the NLS we can find a new solution $(p_1, q_1) = (\mathcal{S}_\alpha(p), \mathcal{S}_\alpha(q))$ by solving a Riccati equation for q_1

$$q_{1,x} = -2q + 2pq_1^2 - 2\alpha q_1$$

and then $p_1 = p^2 q_1 - 2\alpha p - p_x$. Equations (17) can be regarded as an integrable system of differential–difference equations (DΔEs). In variables

$$p_n = \mathcal{S}_\alpha^n(p),\ q_n = \mathcal{S}_\alpha^n(q),\ \alpha_n = \mathcal{S}_\alpha^n(\alpha) \tag{19}$$

it takes the form [17]

$$\begin{array}{rcl} p_{n,x} & = & 2p_{n+1} - 2p_n^2 q_{n+1} + 2\alpha_n p_n, \\ q_{n,x} & = & -2q_{n-1} + 2p_{n-1}q_n^2 - 2\alpha_{n-1}q_n, \end{array} \qquad n \in \mathbb{Z},\quad \alpha_n \in \mathbb{C}. \tag{20}$$

In order to simplify notations we often shall omit n in the lower index for functions depending on the point of the lattice replacing $p_{n\pm1}$ by $p_{\pm1}$, etc. System (20) has a Lax-Darboux representation (often called a semi-discrete Lax representation)

$$L\Psi = 0, \qquad \mathcal{S}_\alpha(\Psi) = M_\alpha \Psi.$$

Its compatibility condition (11) is equivalent to (20). It has an infinite hierarchy of commuting symmetries following from the conditions $\mathcal{S}_\alpha A^k = \mathrm{Ad}_{M_\alpha} A^k$ and local conservation laws. The latter will be shown in the next section.

Automorphism $\mathcal{T}_h$ gives the explicit map for solutions of the NLS:

$$\mathcal{T}_h(p) = p^2 q - \frac{p}{4}\left(\frac{p_x}{p}\right)_x, \qquad \mathcal{T}_h(q) = \frac{1}{p}. \tag{21}$$

In variables $p = \exp\phi,\ \phi_k = \mathcal{T}_h^k(\phi),\ h_k = \mathcal{T}_h^k(h)$ it can be written as

$$\phi_x = -2h, \qquad h_x = 2\exp(\phi_1 - \phi) - 2\exp(\phi - \phi_{-1}) \tag{22}$$

and after elimination of h it takes the form of the Toda lattice:

$$\phi_{xx} = 4\exp(\phi - \phi_{-1}) - 4\exp(\phi_1 - \phi).$$

The DΔEs which follow from $\mathcal{T}_h A^k = \mathrm{Ad}_{N_h} A^k$ are symmetries of (22). For example for $k = 0, 1, 2, 3$ we obtain

$$\left\{\begin{array}{l} \phi_{t_0} = 2, \\ h_{t_0} = 0; \end{array}\right. \qquad \left\{\begin{array}{l} \phi_{t_1} = -2h, \\ h_{t_1} = 2(\mathcal{T}_h - 1)\exp(\phi - \phi_{-1}); \end{array}\right.$$

$$\left\{\begin{array}{l} \phi_{t_2} = 2h^2 - 2(\mathcal{T}_h + 1)\exp(\phi - \phi_{-1}), \\ h_{t_2} = -2(\mathcal{T}_h - 1)(\exp(\phi - \phi_{-1})(h_{-1} + h)); \end{array}\right.$$

$$\left\{\begin{array}{l} \phi_{t_3} = 2\exp(\phi - \phi_{-1})(2h + h_{-1}) + 2\exp(\phi_1 - \phi)(2h + h_1) - 2h^3, \\ \\ h_{t_3} = 2(\mathcal{T}_h - 1)((h_{-1}^2 + h_{-1}h + h^2)\exp(\phi - \phi_{-1}) + \\ \qquad \exp(2\phi - 2\phi_{-1}) + (\mathcal{T}_h + 1)\exp(\phi - \phi_{-2})). \end{array}\right.$$

To define the explicit map (21) we have to consider the localisation of the ring with respect to the element p^{-1}. Then we introduced the exponential function in order to transform the system in the standard well known form of the Toda lattice.

2.3. Bianchi commutativity of Darboux maps and integrable PΔEs

Let us impose the condition that automorphisms corresponding to two Darboux maps $\mathcal{S}_\alpha$ and $\mathcal{S}_\beta$ commute

$$[\mathcal{S}_\alpha,\mathcal{S}_\beta]=0 \quad\Rightarrow\quad \mathcal{S}_\alpha(M_\beta)M_\alpha-\mathcal{S}_\beta(M_\alpha)M_\beta=0. \tag{23}$$

It leads to the Bianchi lattice which can be regarded as a system of partial difference equations (PΔE) on $\mathbb{Z}^2$.

$$\begin{cases} (\mathcal{S}_\beta(p)-\mathcal{S}_\alpha(p))(1+p\mathcal{S}_\beta\mathcal{S}_\alpha(q))-(\alpha-\beta)p=0 \\ (\mathcal{S}_\beta(q)-\mathcal{S}_\alpha(q))(1+p\mathcal{S}_\beta\mathcal{S}_\alpha(q))+(\alpha-\beta)\mathcal{S}_\beta\mathcal{S}_\alpha(q)=0 \end{cases}$$

Denoting $p_{nm}=\mathcal{S}_\alpha^n\mathcal{S}_\beta^m(p), q_{nm}=\mathcal{S}_\alpha^n\mathcal{S}_\beta^m(q)$ we get a quadrilateral system of equations:

$$p_{01} = p_{10}+\frac{\alpha-\beta}{1+p\,q_{11}}p\,,\ q_{01} = q_{10}-\frac{\alpha-\beta}{1+p\,q_{11}}q_{11}\,. \tag{24}$$

DΔEs, which follow from conditions

$$\mathcal{S}_\alpha L=\mathrm{Ad}_{M_\alpha}L,\ \mathcal{S}_\alpha A^k=\mathrm{Ad}_{M_\alpha}A^k \tag{25}$$

are generalised symmetries of (24). Symmetries corresponding to conditions $\mathcal{S}_\beta L = \mathrm{Ad}_{M_\beta}L,\ \mathcal{S}_\beta A^k=\mathrm{Ad}_{M_\beta}A^k$ are equivalent to (25) modulo system (24).

Similarly, the condition $[\mathcal{S}_\alpha,\mathcal{T}_h]=0$ leads to

$$\mathcal{T}_h(M_\alpha)N_h=\mathcal{S}_\alpha(N_h)M_\alpha, \tag{26}$$

which is equivalent to the fully discrete Toda lattice

$$\mathrm{e}^{\phi_{10}-\phi}-\mathrm{e}^{\phi-\phi_{-10}}+\mathrm{e}^{\phi_{-11}-\phi}-\mathrm{e}^{\phi-\phi_{1,-1}}+\alpha-\alpha_{-1}=0, \tag{27}$$

and

$$h=\mathrm{e}^{\phi-\phi_{1,-1}}-\mathrm{e}^{\phi_{10}-\phi}-\alpha,$$

in the variables $\phi_{pq}=\mathcal{T}_h^p\mathcal{S}_\alpha^q\log p$ and $\mathcal{T}_h^k\alpha=\alpha, \mathcal{S}^k\alpha=\alpha_k, \alpha_k\in\mathbb{C}$. The discrete Toda lattice is a difference equation which is defined on a 5–points stencil.

Equations which follow from conditions

$$\mathcal{S}_\alpha L=\mathrm{Ad}_{M_\alpha}L,\ \mathcal{S}_\alpha A^k=\mathrm{Ad}_{M_\alpha}A^k \tag{28}$$

are symmetries of the discrete Toda lattice. For example $\mathcal{S}_\alpha L=\mathrm{Ad}_{M_\alpha}L$ results in

$$\phi_x = -2(\mathrm{e}^{\phi-\phi_{1,-1}}-\mathrm{e}^{\phi_{10}-\phi}-\alpha).$$

Symmetries corresponding to conditions $\mathcal{T}_h L=\mathrm{Ad}_{N_h}L,\ \mathcal{T}_h A^k=\mathrm{Ad}_{N_h}A^k$ are equivalent to (28) modulo system (27).

2.4. Adjacent Lax structure

In Section 2.2 we discussed the problem to find all elementary Darboux matrices corresponding to a given Lax structure. There is another interesting and important problem to find all possible Lax structures associated with a given Darboux matrix. In this section we show that the Darboux matrix M_α (14) corresponding to Lax operators (2) admits an alternative Lax structure with operators $B^k = D_{y_k} - W^k$. We shall assume that $\mathcal{S}_\alpha(\alpha) = \alpha$, i.e. the constant α does not depend on the vertex of the lattice.

We notice that the determinant of the Darboux matrix

$$M_\alpha = \begin{pmatrix} \lambda + p\mathcal{S}_\alpha(q) - \alpha & p \\ \mathcal{S}_\alpha(q) & 1 \end{pmatrix}$$

is $\lambda - \alpha$. Thus at $\lambda = \alpha$ $M_\alpha^0 = M_\alpha|_{\lambda=\alpha}$ has rank 1 and can be represented by a bi-vector

$$M_\alpha^0 = \begin{pmatrix} p \\ 1 \end{pmatrix} \cdot (\mathcal{S}_\alpha(q),\ 1)\,.$$

Let us search for a Lax operator $B_\alpha^1 = D_y - W_\alpha^1$ with a matrix W_α^1 having a simple pole with a residue of rank 1 at $\lambda = \alpha$ and vanishing at $\lambda = \infty$

$$W_\alpha^1 = \frac{\mathcal{W}_\alpha^1}{\lambda - \alpha}.$$

It follows from $\mathcal{S}_\alpha B_\alpha^1 = \mathrm{Ad}_{M_\alpha} B_\alpha^1$ that

$$D_y(M_\alpha) = \mathcal{S}_\alpha(W_\alpha^1)M_\alpha - M_\alpha W_\alpha^1. \tag{29}$$

Taking the residue at $\lambda = \alpha$ we get equation

$$S_\alpha(\mathcal{W}_\alpha^1)M_\alpha^0 = M_\alpha^0\mathcal{W}_\alpha^1$$

which has a unique (up to a scalar constant factor γ) solution

$$\mathcal{W}_\alpha^1 = \frac{\gamma}{1 + \mathcal{S}_\alpha^{-1}(p)\mathcal{S}_\alpha(q)} \begin{pmatrix} \mathcal{S}_\alpha^{-1}(p) \\ 1 \end{pmatrix} \cdot (\mathcal{S}_\alpha(q),\ 1)\,.$$

In what follows we set $\gamma = 1$. Thus

$$W_\alpha^1 = \frac{1}{(\lambda - \alpha)(1 + \mathcal{S}_\alpha^{-1}(p)\mathcal{S}_\alpha(q))} \begin{pmatrix} \mathcal{S}_\alpha^{-1}(p)\mathcal{S}_\alpha(q) & \mathcal{S}_\alpha^{-1}(p) \\ \mathcal{S}_\alpha(q) & 1 \end{pmatrix}.$$

Entries of W_α^1 are not from the differential-difference polynomial ring and localisation of the ring by the element $(1 + \mathcal{S}_\alpha^{-1}(p)\mathcal{S}_\alpha(q))^{-1}$ is required.

With this W_α^1 equation (29) is equivalent to the following evolutionary system of integrable differential-difference equations

$$p_y = -\frac{p_{-1}}{1 + p_{-1}q_1}, \quad q_y = \frac{q_1}{1 + p_{-1}(p)q_1}. \tag{30}$$

Here we use notations introduced in (19). System (30) is a new symmetry of differential-difference system (20) as well as of partial-difference systems (24),(27).

It can be shown that there is an infinite hierarchy of commuting operators

$$B_\alpha^k = D_{y_k} - W_\alpha^k, \qquad W_\alpha^{k+1} = \frac{1}{\lambda - \alpha}(W_\alpha^k + C^k) \tag{31}$$

and C^k is a λ-independent matrix. Using condition $[B_\alpha^1, B_\alpha^2] = 0$ we can find that

$$C^1 = \frac{1}{(1+p_{-1}q_1)^2}\begin{pmatrix} p_{-1}q_{1,y} - p_{-1,y}q_1 & -p_{-1,y} - p_{-1}^2 q_{1,y} \\ q_{1,y} + q_{-1}^2 p_{-1,y} & p_{-1,y}q_1 - p_{-1}q_{1,y} \end{pmatrix}.$$

For matrices W_α^k there exists a recursion (similar to (6)) which enables one to find the infinite hierarchy of operators B_α^k recursively. Operators B_α^k, $k = 1, 2, \ldots$ form the *adjacent* Lax structure.

The partial differential equation which is equivalent to the condition $[B_\alpha^1, B_\alpha^2] = 0$ is of the form

$$\begin{aligned} (p_{-1})_{y_2} &= -(p_{-1})_{yy} + \frac{2q_1((p_{-1})_y)^2}{1+p_{-1}q_1}, \\ (q_1)_{y_2} &= (q_1)_{yy} - \frac{2p_{-1}((q_1)_y)^2}{1+p_{-1}q_1}. \end{aligned} \tag{32}$$

System (30) is a Bäcklund transformation for (32). Equation (32) is well known, it is a Heisenberg model for ferromagnets [18]

$$\mathbf{S}_\tau = \mathbf{S} \times \mathbf{S}_{yy}, \qquad \mathbf{S}^2 = 1$$

after the change of variable $y_2 = i\tau$ and stereographic projection

$$\mathbf{S} = \left(\frac{p_{-1} + q_1}{1 + p_{-1}q_1},\ i\frac{q_1 - p_{-1}}{1 + p_{-1}q_1},\ \frac{p_{-1}q_1 - 1}{1 + p_{-1}q_1} \right).$$

We can use equation (30) to eliminate y derivatives from the Lax operator B_α^2 and partial differential equation (32). The latter will take the form of a differential-difference system

$$\begin{aligned} p_{y_2} &= \frac{p_{-1}^2 q_2(1 + p_{-2}q) - p_{-2}(1 + pq_2)}{(1 + p_{-2}q)(1 + p_{-1}q_1)^2(1 + pq_2)}, \\ q_{y_2} &= \frac{q_2(1 + p_{-2}q) - q_1^2 p_{-2}(1 + pq_2)}{(1 + p_{-2}q)(1 + p_{-1}q_1)^2(1 + pq_2)}. \end{aligned} \tag{33}$$

Equation (33) is a Bäcklund transformation for (32) and a symmetry of systems (30), (20), (24), (27).

The system of partial difference equations (24) is equivalent to the commutativity of Darboux transformations $[\mathcal{S}_\alpha, \mathcal{S}_\beta] = 0$, corresponding to Darboux matrices M_α and M_β, with distinct values of the parameters α and β. With these matrices we associate two Lax operators

$$B_\alpha = D_y - \frac{W_\alpha^1}{\lambda - \alpha}, \quad B_\beta = D_z - \frac{W_\beta^1}{\lambda - \beta}, \qquad \alpha \neq \beta, \tag{34}$$

which coincide with the Lax pair for the principal chiral field model [19]. The compatibility condition $[B_\alpha, B_\beta] = 0$ leads to the system

$$(W^1_\beta)_y = \frac{[W^1_\alpha, W^1_\beta]}{\beta-\alpha}, \qquad (W^1_\alpha)_z = \frac{[W^1_\alpha, W^1_\beta]}{\beta-\alpha},$$

which in variables $p_{nm} = \mathcal{S}^n_\alpha \mathcal{S}^m_\beta(p), q_{nm} = \mathcal{S}^n_\alpha \mathcal{S}^m_\beta(q)$ can be written as

$$\begin{aligned} (p_{-1,0})_z &= \frac{(p_{0,-1}-p_{-1,0})(1+p_{-1,0}q_{0,1})}{(\alpha-\beta)(1+p_{0,-1}q_{0,1})}, \\ (q_{1,0})_z &= \frac{(q_{1,0}-q_{0,1})(1+p_{0,-1}q_{1,0})}{(\alpha-\beta)(1+p_{0,-1}q_{0,1})}, \\ (p_{0,-1})_y &= (p_{-1,0})_z, \qquad (q_{0,1})_y = (q_{1,0})_z. \end{aligned} \tag{35}$$

Finally, let us consider the compatibility condition $[L, B_\alpha] = 0$ for two linear problems

$$L\Psi = 0, \qquad B_\alpha\Psi = 0$$

with the original Lax operator L (2) and operator B_α(34). Vanishing of the commutator at infinity in λ is equivalent to equations (30). Using equations (30) we can express p_{-1} and q_1 as

$$p_{-1} = \frac{1+\sqrt{1+4(p)_y(q)_y}}{2(q)_y}, \quad q_1 = -\frac{1+\sqrt{1+4(p)_y(q)_y}}{2(p)_y}.$$

Of course there is also the second solution with the negative sign at the square root, it can be treated similarly. Then the compatibility conditions are equivalent to the system of partial differential equations

$$(p)_{xy} = 2\alpha(p)_y + 2p\sqrt{1+4(p)_y(q)_y}, \quad (q)_{xy} = -2\alpha(q)_y + 2q\sqrt{1+4(p)_y(q)_y}. \tag{36}$$

The constant α can be removed by a simple change of variables $P = pe^{-2\alpha x},\ Q = qe^{2\alpha x}$. Then the system admits an obvious reduction $P = Q$ to a single hyperbolic equation

$$(P)_{xy} = P\sqrt{1+4((P)_y)^2}.$$

The latter equation is well known in the literature. It can be reduced to the sine-Gordon equation by a differential substitution [20]. The above construction provides us with the Lax representation for the system (36) with the Lax operators L (2) and the second operator

$$B = D_y - \frac{1}{2(\lambda-\alpha)}\begin{pmatrix} \sqrt{1+4(p)_y(q)_y} & -2(p)_y \\ 2(q)_y & -\sqrt{1+4(p)_y(q)_y} \end{pmatrix}. \tag{37}$$

Similarly one can eliminate shifts from operators B^k_α to build up the Lax structure correponding to (36), (37).

3. Formal diagonalisation of the Lax-Darboux Scheme

In this section we show that the Lax operators and corresponding Darboux matrices can be simultaneously formally diagonalised. The resulting objects will be presented by formal Laurent expansions at poles of the chosen Lax operator. It will enable us to find recursively local conservation laws for corresponding partial differential, differential-difference and partial difference equations simultaneously. Our aim is to show that for evolutionary equations from the same Lax-Darboux scheme with Lax operators $L, A^k,\ k = 0, 1, ...$ and Darboux maps $\mathcal{S}_i,\ i = 1, 2, ...$ there is an infinite sequence of common local conservation laws with densities ρ_n, r_n^i and fluxes σ_n^k. That is,

1. PDE's corresponding to the Lax structure $[L, A^k] = 0$ possess conservation lows
$$D_{t_k}\rho_n = D_x\sigma_n^k, \qquad k = 0, 1, \dots\ .$$
2. For differential-difference equations originating from the conditions $\mathcal{S}_i L = \mathrm{Ad}_{M_i} L$ and $\mathcal{S}_i A^k = \mathrm{Ad}_{M_i} A^k$ there are conservation laws of the form
$$D_x r_n^i = (\mathcal{S}_i - 1)\rho_n, \qquad D_{t_k} r_n^i = (\mathcal{S}_i - 1)\sigma_n^k$$
with the same ρ_n and σ_n^k as above, modulo equation $\mathcal{S}_i L = \mathrm{Ad}_{M_i} L$.
3. For partial difference equations, corresponding to the Bianchi lattice $[\mathcal{S}_i, \mathcal{S}_j] = 0$ the corresponding sequence of the conservation laws are:
$$(\mathcal{S}_j - 1)r_n^i = (\mathcal{S}_i - 1)r_n^j.$$

We shall demonstrate it on the examples described in the previous section, associated with the Lax-Darboux scheme for the nonlinear Schrödinger equation as well as with the adjacent Lax structures considered in Section 2.4. A generalisation of this approach to other Lax-Darboux schemes (or their parts) often is rather straightforward and it will be discussed at the end of this Section.

3.1. Formal diagonalisation of the Lax structure for NLS (L, A^k)

In the Lax operator L (2) matrix U has a simple pole in λ at infinity with the coefficient J which is diagonal (3). The matrices V^k in operators $A^k = D_{t_k} - V^k$ are differential polynomials in variables p, q and their x derivatives with complex coefficients. The leading (in λ) coefficient is also diagonal and is equal to $\lambda^k J$. By *local* functions (in this case) we shall understand elements of the differential polynomial ring $\mathcal{R}_x = [\mathbb{C}; p, q; D_x]$.

Let us consider endomorphism ad_J of the linear space $\mathfrak{M} = \mathrm{Mat}_{2\times 2}(\mathcal{R}_x)$ of 2×2 matrices with entries from $\mathcal{R}_x$
$$\mathrm{ad}_J : \mathfrak{M} \mapsto \mathfrak{M}, \quad \mathrm{ad}_J(a) = Ja - aJ, \quad a \in \mathfrak{M}.$$
The kernel of ad_J is the subspace of diagonal matrices, the image space of ad_J is a subspace of off-diagonal matrices. Thus
$$\mathfrak{M} = \mathfrak{M}_\parallel \oplus \mathfrak{M}_\perp, \qquad \mathfrak{M}_\parallel = \mathrm{Ker}\,\mathrm{ad}_J,\ \mathfrak{M}_\perp = \mathrm{Im}\,\mathrm{ad}_J.$$

In $\mathfrak{M}_\perp$ the endomorphism ad_J is invertible

$$\mathrm{ad}_J^{-1} : \mathfrak{M}_\perp \mapsto \mathfrak{M}_\perp, \quad \mathrm{ad}_J^{-1} a = \frac{1}{4}\mathrm{ad}_J a, \qquad \forall a \in \mathfrak{M}_\perp.$$

In the space $\mathfrak{M}$ it is convenient to introduce two projectors

$$\pi_\perp = \frac{1}{4}\mathrm{ad}_J^2, \quad \pi_\parallel = id - \pi_\perp$$

where id is the identity map. They are projectors on the off-diagonal and diagonal part of a matrix respectively

$$\pi_\perp \mathfrak{M} = \mathfrak{M}_\perp, \qquad \pi_\parallel \mathfrak{M} = \mathfrak{M}_\parallel.$$

We shall use a simplified version of the Drinfeld-Sokolov Lemma, which they have formulated and proved in a rather general setting [7].

Lemma 1. *For linear operator L (2) there exists a unique formal series*

$$Q = I + \lambda^{-1}Q_1 + \lambda^{-2}Q_2 + \lambda^{-3}Q_3 + \cdots, \qquad Q_k \in \mathfrak{M}_\perp \tag{38}$$

such that

$$\mathcal{L} = Q^{-1}LQ = D_x - \lambda J - \mathcal{U}_0 - \lambda^{-1}\mathcal{U}_1 - \lambda^{-2}\mathcal{U}_2 - \cdots, \qquad \mathcal{U}_k \in \mathfrak{M}_\parallel. \tag{39}$$

The coefficients Q_k can be found recursively

$$Q_1 = -\frac{1}{4}\mathrm{ad}_J U, \qquad Q_{k+1} = \frac{1}{4}\mathrm{ad}_J\left(D_x(Q_k) + \sum_{p=1,q=1}^{p+q=k} Q_p U Q_q\right), \tag{40}$$

and

$$\mathcal{U}_0 = 0, \qquad \mathcal{U}_k = UQ_k \ .$$

Proof. Let us substitute $L, \mathcal{L}$ and Q in

$$C = LQ - Q\mathcal{L} = C_0 + \lambda^{-1}C_1 + \lambda^{-2}C_2 + \cdots \ .$$

The condition that the formal series C should vanish provides us with a sequence of equations to determine the coefficients $\mathcal{U}_k, Q_k$. The linear in λ term vanishes automatically. The coefficient at λ^0 is

$$C_0 = [J, Q_1] + U - \mathcal{U}_0.$$

Applying projectors $\pi_\parallel$ and $\pi_\perp$ to C_0 we find that

$$\pi_\parallel([J, Q_1] + U - \mathcal{U}_0) = -\mathcal{U}_0 = 0, \qquad \pi_\perp([J, Q_1] + U - \mathcal{U}_0) = [J, Q_1] + U = 0.$$

Thus $\mathcal{U}_0 = 0$ and $Q_1 = -\frac{1}{4}\mathrm{ad}_J U$. The coefficient at λ^{-k} is

$$C_k = \mathrm{ad}_J Q_{k+1} - D_x(Q_k) + UQ_k - \mathcal{U}_k - \sum_{p=1,q=1}^{p+q=k} Q_p \mathcal{U}_q.$$

Therefore

$$\pi_{\|}(C_k) = UQ_k - \mathcal{U}_k = 0 \ \Rightarrow \mathcal{U}_k = UQ_k,$$

$$\pi_{\perp}(C_k) = \mathrm{ad}_J Q_{k+1} - D_x(Q_k) - \sum_{p=1,q=1}^{p+q=k} Q_p \mathcal{U}_q = 0 \ .$$

Thus the coefficients $Q_n, \mathcal{U}_n$ can be found recursively:

$$Q_1 = -\frac{1}{4}\mathrm{ad}_J U, \ Q_{k+1} = \frac{1}{4}\mathrm{ad}_J \left(D_x(Q_k) + \sum_{p=1,q=1}^{p+q=k} Q_p U Q_q \right), \ \mathcal{U}_k = UQ_k.$$

Note that Q_k and $\mathcal{U}_k$ are all local, i.e. expressed in terms of differential polynomials. $\square$

We are going to show that Q diagonalises the whole Lax-Darboux scheme, i.e. diagonalises the operators A^k and the Darboux matrices M_α (corresponding to $\mathcal{S}_\alpha$). It is well known that all operators A^k in the Lax structure become diagonal and lead to local conservation laws for the corresponding partial differential equations and their symmetries.

Proposition 1. *Let* $[L, A^k] = 0$, *where*

$$A^k = D_{t_k} - \lambda^k J - \lambda^{k-1} V^k_{1-k} - \lambda^{k-2} V^k_{2-k} - \cdots \ .$$

Then

$$\mathcal{A}^k = Q^{-1} A^k Q = D_{t_k} - \lambda^k J - \lambda^{k-1} \mathcal{V}^k_{1-k} - \lambda^{k-2} \mathcal{V}^k_{2-k} + \cdots \tag{41}$$

has diagonal coefficients $\mathcal{V}^k_s \in \mathfrak{M}_{\|}$, $s = k-1, k-2, \dots$.

Proof. If $[L, A^k] = 0$, then $[\mathcal{L}, \mathcal{A}^k] = 0$ where $\mathcal{L} = Q^{-1}LQ$ (39) and $\mathcal{A}^k = Q^{-1}A^kQ$ (41). Using induction we show that all coefficients of the formal series $\mathcal{A}^k$ are diagonal. The leading coefficient of the series $\lambda^k J$ is diagonal. Let us assume that coefficients $\mathcal{V}^k_{1-k}, \mathcal{V}^k_{2-k}, \dots, \mathcal{V}^k_{n-k}$ are diagonal. Then the leading term of $\pi_\perp[\mathcal{L}, \mathcal{A}^k]$ is equal to $\lambda^{k-n}\mathrm{ad}_J \mathcal{V}^k_{n+1-k}$. It should vanish and thus $\mathcal{V}^k_{n+1-k} \in \mathfrak{M}_{\|}$. $\square$

Corollary 1. *The following ystems of partial differential equations*

$$(p_{t_k} = f^k, \ q_{t_k} = g^k) \ \Leftrightarrow \ [L, A^k] = 0$$

have an infinite hierarchy of common conservation laws

$$(\mathcal{U}_n)_{t_k} = D_x(\mathcal{V}^k_n), \qquad n = 1, 2, \dots \ ,$$

Moreover, $D_x \mathcal{V}^k_m = 0$, *for* $m = 1-k, 2-k, \dots, 1, 0$.

Proof. From $[L, A^k] = 0$ it follows that $[\mathcal{L}, \mathcal{A}^k] = 0$ which leads to

$$\sum_{n=1}^{\infty} \lambda^{-n} (\mathcal{U}_n)_{t_k} - \sum_{n=1-k}^{\infty} \lambda^{-n} (\mathcal{V}^k_n)_x = 0.$$

Vanishing the coefficients at each power λ^n proves the statement. $\square$

It is easy to show that differential polynomials $\mathrm{Tr}\,\mathcal{U}_n \in \mathrm{Im}\, D_x$ and thus they correspond to trivial densities. Let us take the matrix entry $(\mathcal{U}_n)_{2,2},\ (\mathcal{V}_n^k)_{2,2}$ to define

$$\rho_n = (\mathcal{U}_n)_{2,2}, \qquad \sigma_n^k = (\mathcal{V}_n^k)_{2,2}.$$

It can be shown that the corresponding densities are all non-trivial.

Example. Taking L corresponding to the NLS (2) we find that

$$Q_1 = \begin{pmatrix} 0 & -p \\ q & 0 \end{pmatrix}, \qquad \mathcal{U}_1 = 2pq \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

$$Q_2 = -\tfrac{1}{2} \begin{pmatrix} 0 & p_x \\ q_x & 0 \end{pmatrix}, \qquad \mathcal{U}_2 = -\begin{pmatrix} pq_x & 0 \\ 0 & qp_x \end{pmatrix},$$

$$Q_3 = \tfrac{1}{4} \begin{pmatrix} 0 & -p_{xx} + 4p^2q \\ q_{xx} - q^2p & 0 \end{pmatrix}, \quad \mathcal{U}_3 = \tfrac{1}{2} \begin{pmatrix} pq_{xx} - 4p^2q^2 & 0 \\ 0 & 4p^2q^2 - qp_{xx} \end{pmatrix}, \ldots$$

and

$$\rho_1 = -2pq,\ \sigma_1^2 = q_xp - p_xq,\ \sigma_1^3 = \tfrac{1}{2}(p_xq_x - pq_{xx} - qp_{xx}) + 6p^2q^2\ \ldots,$$

$$\rho_2 = -qp_x,\ \sigma_2^2 = \tfrac{1}{2}(p_xq_x - qp_{xx}) + 2p^2q^2,\ \sigma_2^3 = \tfrac{1}{4}(p_{xx}q_x - qp_{xxx}) + 4pq^2p_x\ \ldots,$$

$$\rho_3 = 2p^2q^2 - \tfrac{1}{2}qp_{xx},\ \rho_4 = pq(pq_x + 4qp_x)) - \tfrac{1}{4}qp_{xxx}\ \ldots.$$

3.2. Formal diagonalisation of the Darboux matrices M_α, N_h

The diagonalising transformation Q can be extended to the Darboux matrices M_α, M_β and N_h. We substitute $L = Q\mathcal{L}Q^{-1}$ in $\mathcal{S}_\alpha(L) = M_\alpha L M_\alpha^{-1}$ to obtain

$$\mathcal{S}_\alpha(\mathcal{L}) = \mathcal{M}_\alpha \mathcal{L} \mathcal{M}_\alpha^{-1}, \qquad \mathcal{M}_\alpha = \mathcal{S}_\alpha(Q)^{-1} M_\alpha Q. \tag{42}$$

Similarly we obtain

$$\mathcal{T}_h(\mathcal{L}) = \mathcal{N}_h \mathcal{L} \mathcal{N}_h^{-1}, \qquad \mathcal{N}_h = \mathcal{T}_h(Q)^{-1} N Q.$$

These equations can be rewritten in the form

$$D_x(\mathcal{M}_\alpha) = \mathcal{S}_\alpha(\mathcal{L})\mathcal{M}_\alpha - \mathcal{M}_\alpha \mathcal{L}\,, \tag{43}$$
$$D_x(\mathcal{N}_h) = \mathcal{T}_h(\mathcal{L})\mathcal{N}_h - \mathcal{N}_h \mathcal{L}\,. \tag{44}$$

Proposition 2. *The coefficients $\mathcal{M}_\alpha^k,\ \mathcal{N}_h^k$ of the formal series*

$$\mathcal{M}_\alpha = \mathcal{S}_\alpha(Q)^{-1} M_\alpha Q = \lambda \mathcal{M}_\alpha^{-1} + \mathcal{M}_\alpha^0 + \lambda^{-1}\mathcal{M}_\alpha^1 + \lambda^{-2}\mathcal{M}_\alpha^2 + \cdots$$

and

$$\mathcal{N}_h = \mathcal{T}_h(Q)^{-1} N Q = \lambda \mathcal{N}_h^{-1} + \mathcal{N}_h^0 + \lambda^{-1}\mathcal{N}_h^1 + \lambda^{-2}\mathcal{N}_h^2 + \cdots$$

are diagonal matrices.

Proof. We prove it by induction. The leading coefficients $\mathcal{M}_\alpha^{-1} = \mathcal{N}_h{-}1$ are diagonal matrices with $(1,0)$ on the diagonal. Let us assume that the coefficients $\mathcal{M}_\alpha^0, \dots, \mathcal{M}_\alpha^n$ are diagonal. Taking the coefficient c_n at λ^{-n} in $D_x(\mathcal{M}_\alpha) - \mathcal{S}_\alpha(\mathcal{L})\mathcal{M}_\alpha + \mathcal{M}_\alpha\mathcal{L}$ we obtain

$$c_n = D_x(\mathcal{M}_\alpha^n) - [J, \mathcal{M}_\alpha^{n+1}] - \sum_{k=-1}^{n} \mathcal{S}_\alpha(\mathcal{U}_{n-k})\mathcal{M}_\alpha^k + \mathcal{M}_\alpha^k\mathcal{U}_{n-k} = 0.$$

Projection $\pi_\perp(c_n) = -[J, \mathcal{M}_\alpha^{n+1}] = 0$, which implies that $\mathcal{M}_\alpha^{n+1}$ is diagonal. The proof is similar for the coefficients of $\mathcal{N}_h$. □

Equation (43) can be written in the form

$$D_x(\log \mathcal{M}_\alpha) = (\mathcal{S}_\alpha - I)\mathcal{L}\,,$$

since all matrices in (43) are diagonal. Thus $\log \mathcal{M}_\alpha$ is a generating function for local conservation laws for the differential-difference equation (20):

$$\log(\mathcal{M}_\alpha)_{2,2} = \frac{r_\alpha^1}{\lambda} + \frac{r_\alpha^2}{\lambda^2} + \frac{r_\alpha^3}{\lambda^3} + \cdots$$

$$D_x(r_\alpha^k) = \mathcal{S}_\alpha(\rho_k) - \rho_k.$$

It follows from (12) and Proposition 1 that

$$D_{t_k}(r_\alpha^n) = \mathcal{S}_\alpha(\sigma_k^n) - \sigma_k^n.$$

Moreover, it follows from (23) that

$$\mathcal{S}_\alpha(\mathcal{M}_\beta)\mathcal{M}_\alpha = \mathcal{S}_\beta(\mathcal{M}_\alpha)\mathcal{M}_\beta.$$

Therefore

$$(\mathcal{S}_\alpha - I)\log \mathcal{M}_\beta = (\mathcal{S}_\beta - I)\log \mathcal{M}_\alpha$$

and thus

$$(\mathcal{S}_\alpha - I)r_\beta^k = (\mathcal{S}_\beta - I)r_\alpha^k, \qquad k = 1, 2, \dots .$$

Similarly, from (26) it follows that $\mathcal{T}_h(\mathcal{M}_\alpha)\mathcal{N}_h = \mathcal{S}_\alpha(\mathcal{N}_h)\mathcal{M}_\alpha$ and thus

$$(\mathcal{S}_\alpha - I)r^k = (\mathcal{T}_h - I)r_\alpha^k, \qquad \log(\mathcal{N}_h)_{2,2} = \frac{r^1}{\lambda} + \frac{r^2}{\lambda^2} + \frac{r^3}{\lambda^3} + \cdots .$$

Example. Using equations (17), (18) corresponding to $\mathcal{S}_\alpha L = \mathrm{Ad}_{M_\alpha} L, \mathcal{T}_h L = \mathrm{Ad}_N L$ for elimination of all x-derivatives we get

$$Q = I + \lambda^{-1}\begin{pmatrix} 0 & -p \\ q & 0 \end{pmatrix} + \lambda^{-2}\begin{pmatrix} 0 & -\alpha p - \mathcal{S}_\alpha(p) + p^2\mathcal{S}_\alpha(q) \\ \mathcal{S}_\alpha^{-1}(q) + \alpha q - \mathcal{S}_\alpha^{-1}(p)q^2 & 0 \end{pmatrix} + \cdots =$$

$$I + \lambda^{-1}\begin{pmatrix} 0 & -p \\ q & 0 \end{pmatrix} + \lambda^{-2}\begin{pmatrix} 0 & hp \\ -\mathcal{T}_h^{-1}(hp^{-1}) & 0 \end{pmatrix} +$$

$$+\lambda^{-3}\begin{pmatrix} 0 & -h^2p - \mathcal{T}_h(p) \\ \mathcal{T}_h^{-1}(h^2p^{-1}) - \mathcal{T}_h^{-2}(p^{-1}) & 0 \end{pmatrix} + \cdots$$

and

$$\mathcal{M}_\alpha = \lambda \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} p\mathcal{S}_\alpha(q) - \alpha & 0 \\ 0 & 1 \end{pmatrix} + \lambda^{-1} \begin{pmatrix} pq & 0 \\ 0 & -p\mathcal{S}_\alpha(q) \end{pmatrix} + \cdots$$

$$\mathcal{N}_h = \lambda \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} h & 0 \\ 0 & 0 \end{pmatrix} + \lambda^{-1} \begin{pmatrix} p\mathcal{T}_h^{-1}(p^{-1}) & 0 \\ 0 & -1 \end{pmatrix} + \lambda^{-2} \begin{pmatrix} -p\mathcal{T}_h^{-1}(hp^{-1}) & 0 \\ 0 & h \end{pmatrix} + \cdots$$

Thus

$$\rho_1 = -2pq = -2\exp(\phi - \mathcal{T}_h^{-1}\phi), \tag{45}$$

$$\rho_2 = -qp_x = -2\left(\alpha pq + \mathcal{S}_\alpha(p)q - p^2 q\mathcal{S}_\alpha(q)\right) = 2\exp(\phi - \mathcal{T}_h^{-1}\phi)h \tag{46}$$

$$\sigma_2^1 = pq_x - qp_x, \quad \sigma_2^2 = 4p^2q^2 + \frac{1}{2}(p_xq_x - qp_{xx}), \tag{47}$$

$$r_\alpha^1 = -p\mathcal{S}_\alpha(q), \quad r_\alpha^2 = \frac{1}{2}p^2q^2 - \alpha p\mathcal{S}_\alpha(q) - \mathcal{S}_\alpha(pq), \tag{48}$$

$$r^1 = -h, \quad r^2 = \frac{1}{2}h^2 - \exp(\mathcal{T}_h(\phi) - \phi), \quad r^3 = -\frac{1}{3}h^3 + \exp(\mathcal{T}_h(\phi) - \phi)(h + \mathcal{T}_h(h)), \ldots$$

In applications to differential-difference equations one also need to eliminate x–derivatives from σ_2^1, σ_2^2 using equations (17), (18).

3.3. Diagonalisation of adjacent Lax structure

It is easy to justify that the transformation (38), which formally diagonalises the Lax operator L (Lemma 1) and operators A^k, also diagonalises the operators B_α^k (31) associated with the adjacent Lax structure. For example,

$$Q^{-1}B_\alpha^1 Q = \frac{\lambda^{-1}}{1 + p_{-1}q_1} \begin{pmatrix} p_{-1}q_1 & 0 \\ 0 & 1 \end{pmatrix} + \frac{\lambda^{-2}}{1 + p_{-1}q_1} \begin{pmatrix} p_{-1}(q + \alpha q_1) & 0 \\ 0 & \alpha - pq_1 \end{pmatrix} + \cdots .$$

Thus, the coefficients $\hat{\sigma}_\alpha^k$

$$\hat{\sigma}_\alpha^1 = \frac{1}{1 + p_{-1}q_1} = 1 - \sqrt{1 + 4p_yq_y},$$
$$\hat{\sigma}_\alpha^2 = \frac{\alpha + pq_1}{1 + p_{-1}q_1} = \alpha(1 - \sqrt{1 + 4p_yq_y}) + pq_y, \ \ldots$$

in the expansion

$$\left(Q^{-1}B_\alpha^1 Q\right)_{2,2} = \hat{\sigma}_\alpha^1\lambda^{-1} + \hat{\sigma}_\alpha^2\lambda^{-2} + \hat{\sigma}_\alpha^3\lambda^{-3} + \cdots$$

are fluxes for the local conservation laws of (30)

$$D_y(r_\alpha^1) = (\mathcal{S}_\alpha - 1)\hat{\sigma}_\alpha^1, \qquad D_y(r_\alpha^2) = (\mathcal{S}_\alpha - 1)\hat{\sigma}_\alpha^2, \ldots$$

andfor (36)

$$D_y(\rho_1) = D_x(\hat{\sigma}_\alpha^1), \qquad D_y(\rho_2) = D_x(\hat{\sigma}_\alpha^2), \ldots$$

where r_α^1, r_α^2 and ρ_1, ρ_1 are given in (48) and (45),(46) respectively.

Lax operator B_α^1 has a pole at $\lambda = \alpha$ and we can diagonalise it around this pole. It is convenient to introduce a local parameter $\mu = (\lambda - \alpha)^{-1}$ and diagonalise the coefficient at the pole by the gauge transformation

$$\hat{B}_\alpha^1 = T_0^{-1}B_\alpha^1 T_0 = D_y - \mu J_1 + \hat{W}, \tag{49}$$

where

$$T_0 = \begin{pmatrix} p_{-1} & -1 \\ 1 & q_1 \end{pmatrix}, \quad J_1 = \frac{1}{2}(I+J) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \; \hat{W} = \begin{pmatrix} -q_1 p_{-1,y} & -q_{1,y} \\ p_{-1,y} & p_{-1}q_{1,y} \end{pmatrix}.$$

Proposition 3. *Transformation $T^{-1}\hat{B}T := \mathcal{B}$ brings operator $\hat{B}$ (49) to a diagonal form*

$$\mathcal{B} = D_y - \mu J_1 - \mathcal{W}_0 - \mu^{-1}\mathcal{W}_1 - \mu^{-2}\mathcal{W}_2 + \cdots, \qquad \mathcal{W}_k = \pi_{\|}(\mathcal{W}_k),$$

Where

$$\mathcal{W}_0 = \pi_{\|}(\hat{W}), \qquad \mathcal{W}_k = \pi_{\perp}(\hat{W})T_k$$
$$T = I + \mu^{-1}T_1 + \mu^{-2}T_2 + \cdots$$

and off–diagonal coefficients T_k can be found recursively

$$T_1 = -\frac{1}{2}\mathrm{ad}_J(\hat{W}),$$
$$T_{k+1} = \frac{1}{2}\mathrm{ad}_J\left(T_{k,y} - \pi_{\|}(\hat{W})T_k + T_k\pi_{\|}(\hat{W}) + \sum_{s=1}^{k-1} T_{k-s}\pi_{\perp}(\hat{W})T_s\right).$$

We omit the proof since it is very similar to the proof of Lemma 1.

The same transformation $\mathcal{L} = (T_0T)^{-1}LT_0T$ brings the Lax operator L to a diagonal form (this diagonalisation is different from the one given in Lemma 1). The coefficients $\varrho_k = (\mathcal{W}_k)_{2,2}$ of the expansion

$$\varrho_0 = -\frac{p_{-1}q_{1,y}}{1+p_{-1}q_1}, \qquad \varrho_1 = -\frac{p_{-1,y}q_{1,y}}{(1+p_{-1}q_1)^2}, \ldots$$

are densities of the conservation laws for the Heisenberg hierarchy (32), principal chiral field model (35) and system (36).

It can be easily shown that transformation $\mathcal{M}_\alpha = \mathcal{S}_\alpha(T_0T)^{-1}M_\alpha T_0 T$ brings the Darboux matrix M_α in a diagonal form $\mathcal{M}_\alpha$. To apply the transformation to M_α we need to eliminate the y–derivatives from the coefficients T_k using equation (30).

There is a direct way to diagonalise the Darboux matrix M_α. Matrix M_α has two points on the Riemann sphere, where the leading coefficient (in the local parameter) is singular. Indeed, at $\lambda = \infty$ and $\lambda = \alpha$ the leading coefficients are

$$\lambda\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} pq_1 & p \\ q_1 & 1 \end{pmatrix}$$

respectively. Let us diagonalise the Darboux matrix at $\lambda = \alpha$ without using the result Proposition 3. Namely, we can find coefficients of a formal series

$$T = I + \mu^{-1}T_1 + \mu^{-2}T_2 + \mu^{-3}T_3 + \cdots, \qquad T_k = \pi_{\perp}(T_k)$$

such that the coefficients $\tilde{\mathcal{M}}_k$ in

$$\tilde{\mathcal{M}}_\alpha = \mathcal{S}_\alpha(T)^{-1}\tilde{M}_\alpha T = \tilde{\mathcal{M}}_0 + \mu^{-1}\tilde{\mathcal{M}}_1 + \mu^{-2}\tilde{\mathcal{M}}_2 + \cdots, \qquad \tilde{\mathcal{M}}_k = \pi_{\|}(\tilde{\mathcal{M}}_k) \tag{50}$$

where

$$\tilde{M}_\alpha = \mathcal{S}_\alpha(T_0)^{-1}M_\alpha T_0 = \tilde{M}_0 + \mu^{-1}\tilde{M}_1,$$
$$\tilde{M}_0 = \begin{pmatrix} 1+p_{-1}q_1 & 0 \\ 0 & 0 \end{pmatrix}, \; \tilde{M}_1 = \frac{1}{1+pq_2}\begin{pmatrix} p_{-1}q_2 & -q_2 \\ -p_{-1} & 1 \end{pmatrix}.$$

Proposition 4. *The coefficients $T_k,\ T_k = \pi_\perp(T_k)$ such that the coefficients $\tilde{\mathcal{M}}_0 = \tilde{\mathcal{M}}_0,\ \tilde{\mathcal{M}}_{k+1} = \pi_\perp(\tilde{M}_1)T_k$ are diagonal can be found recursively*

$$T_1 = \frac{1}{(1+p_{-1}q_1)(1+pq_2)}\begin{pmatrix} 0 & q_2 \\ -p_{-1} & 0 \end{pmatrix},$$

$$\tilde{\mathcal{M}}_0 T_{k+1} - \mathcal{S}_\alpha(T_{k+1})\tilde{\mathcal{M}}_0 + \pi_\|(\tilde{M}_1)T_k - \sum_{s=1}^{k}\mathcal{S}_\alpha(T_s)\pi_\perp(\tilde{M}_1)T_{k-s} = 0.$$

The proof is straightforward. What is important here is that in the recursion we do not need to solve difference equations since $\operatorname{rank}\tilde{\mathcal{M}}_0 = 1$ and $\operatorname{Ker}\tilde{\mathcal{M}}_0 \bigoplus \operatorname{Im}\tilde{\mathcal{M}}_0 = \mathbb{C}^2$. Therefore all entries of T_k and $\tilde{\mathcal{M}}_k$ are elements of the difference ring $[\mathbb{C}; p, q, (1+p_{-1}q_1)^1; \mathcal{S}_\alpha]$, i.e difference polynomials of variables $p, q, (1+p_{-1}q_1)^{-1}$ and their $\mathcal{S}_\alpha^k,\ k \in \mathbb{Z}$ shifts with complex coefficients.

It follows from Proposition 4 that

$$\tilde{\mathcal{M}}_\alpha = \begin{pmatrix} 1+p_{-1}q_1 & 0 \\ 0 & 0 \end{pmatrix} + \frac{\mu^{-1}}{1+pq_2}\begin{pmatrix} p_{-1}q_2 & 0 \\ 0 & 1 \end{pmatrix} + \frac{\mu^{-2}}{1+p_{-1}q_1}\begin{pmatrix} \frac{p_{-2}q_2}{1+p_{-1}q} & 0 \\ 0 & -\frac{p_{-1}q_2}{1+pq_2} \end{pmatrix} + \cdots$$

and thus the coefficients $\tilde{r}_{k\alpha}$

$$\tilde{r^0}_\alpha = -\log(1+pq_2), \quad \tilde{r^1}_\alpha = \frac{p_{-1}q_2}{(1+pq_2)(1+p_{-1}q_1)}, \ldots$$

in the expansion of $(\tilde{\mathcal{M}}_\alpha)_{2,2} = -\log(\mu) + \tilde{r^0}_\alpha + \mu^{-1}\tilde{r^1}_\alpha + \cdots$ are new densities of local conservation laws for differential difference equations (20), (30) and partial difference equations (24), (27).

It is obvious that the transformation constructed in Proposition 3 and in Proposition 4 coincide modulo equation (30) and these two approaches are equivalent.

3.4. Summary

In this paper we have presented the concept of Lax-Darboux scheme and illustrated it on the example of the NLS equation. From the differential-difference algebra point of view the scheme can be described as follows. The base object is a differential-difference ring polynomials $\mathcal{R} = [\mathbb{C}; \mathbf{u}; D_{x_1}, D_{x_2}, \ldots; \mathcal{S}_1, \mathcal{S}_2, \ldots]$ of a (vector) variable $\mathbf{u} = u^1, \ldots, u^M$, its derivatives and shifts $D_{x_1}^{n_1} \cdots D_{x_m}^{n_m}\mathcal{S}_1^{s_1} \ldots \mathcal{S}_p^{s_p}\mathbf{u}$, equipped with a set of commuting derivations $D_{x_k},\ k = 1, 2, \ldots$ and commuting automorphisms $\mathcal{S}_i,\ i = 1, 2 \ldots$. To each D_{x_k} we associate a Lax operator of the form $L^k = D_{x_k} - U^k$, where U^k is $N \times N$ matrix with entries belonging to $\mathcal{R}(\lambda)$, i.e. are rational functions of a spectral parameter λ with coefficients from $\mathcal{R}$. With each automorphism $\mathcal{S}_i$ we associate a Darboux $N \times N$ matrix M^i with entries from $\mathcal{R}(\lambda)$. Then the system of Lax-Darboux equations we identify with the differential-difference ideal

$$\mathcal{I} = \langle [L^i, L^j],\ \mathcal{S}_i(L^j) - \mathrm{Ad}_{M^i}(L^j),\ \mathcal{S}_i(M^j)M^i - \mathcal{S}_j(M^j)M^j \rangle \ \subset \mathcal{R}$$

and consider a quotient ring $\mathcal{R}_\mathcal{I} = \mathcal{R}/\mathcal{I}$. In this setup a statement that two expressions are equal modulo equations simply means that these two expressions are equal as elements of $\mathcal{R}_\mathcal{I}$.

We can formally diagonalise (or bring to a block-diagonal form) simultaneously all matrices U^i, M^j near singular points in λ and generate infinite sequences ρ^i_k, r^j_k, $k = 1, 2, \dots$ such that in $\mathcal{R}_\mathcal{I}$ they satisfy relations

$$\begin{aligned} &D_{x_n}\rho^j_k = D_{x_j}\rho^n_k,\\ &D_{x_n}r^j_k = (\mathcal{S}_j - 1)\rho^n_k,\\ &(\mathcal{S}_i - 1)r^j_k = (\mathcal{S}_j - 1)r^i_k. \end{aligned}$$

These relations can be regarded as a sequences of local conservation laws for partial differential, differential difference and partial difference equations. The Lax-Darboux scheme can be generalised to the case when the derivations are not commuting, but such generalisatins are beyond of the scope of this paper.

In the case of the NLS equation the elements of the Lax-Darboux scheme are:

- The Lax structure, i.e. Lax operators L, A^k such that the commutativity conditions $[L, A^k] = [A^k, A^p] = 0$ are equivalent to a system of integrable partial differential equations and its symmetries (5).

- Darboux transformations $\mathcal{S}_\alpha, \mathcal{T}_h$ with Darboux matrices M_α (14) and N_h (15) respectively. The compatibility conditions $\mathcal{S}_\alpha L = \mathrm{Ad}_{M^\alpha} L$, $\mathcal{T}_h L = \mathrm{Ad}_{N_h} L$ and $\mathcal{S}_\alpha A^k = \mathrm{Ad}_{M^\alpha} A^k$, $\mathcal{T}_h A^k = \mathrm{Ad}_{N_h} A^k$ result in Bäcklund transformations of the above integrable system and its symmetries (20), (21). Bäcklund transformations also can be regarded as integrable differential-difference equations in their own right.

- The Bianchi lattices, which follow from the commutativity conditions for pairs of Darboux transformations result in integrable systems of partial difference equations (24), (27). The mentioned above differential-difference equations (20), (21) are symmetries of these systems.

- There is an adjacent Lax structure (corresponding to operators B^k_α (31)) sharing the same Darboux matrix M_α and resulting in the differential-difference integrable system (30). The commutativity condition $[B^1_\alpha, B^k_\alpha] = 0$ results in the hierarchy of the Heisenberg model (32). The commutativity condition $[B^1_\alpha, B^1_\beta] = 0$ is equivalent to the principal chiral field model (35), so that the hierarchy of the Heisenberg equation is a hierarchy of symmetries for (35). Equation $[L, B^k_\alpha] = 0$ provide us with a hierarchy of symmetries for system (36). Integrable differential-difference systems of equations arising from the conditions $\mathcal{S}_\alpha(B^k_\alpha) = \mathrm{Ad}_{M_\alpha}(B^k_\alpha)$ (such as (30) and (33)) are Bäcklund transformations for the above listed hierarchies and symmetries for differential-difference equations (20), (21) and partial difference equations (24), (27).

We have shown that there is a formal diagonalistaion of the Lax-Darboux scheme, i.e. a transformation (in the form of a formal series in the spectral parameter) which diagonalises simultaneously the Lax structure, associated Darboux transformations and adjacent Lax structures. The diagonalised Lax (and adjacent Lax) operators and logarithms of the diagonalised Darboux matrices are generating functions of local conservation laws (both the densities and fluxes) for related partial differential, differential-difference and partial difference equations, which are neatly related to each other. Moreover, there may exist several different diagonalisations, which lead to adjacent hierarchies of local conservation laws for equations corresponding to the Lax-Darboux scheme.

Acknowledgements

I would like to thank participants of my lecture courses, seminars and conferences, where several parts of this study were presented, for their useful comments. In particular I am grateful to G. Berkeley, R.N. Garifullin, S. Konstantinou-Rizos, V.V. Sokolov, R.I. Yamilov and J.P. Wang for discussions and advice. I am gratefully acknowledge support from the Leverhulme Trust.

References

[1] V. B. Matveev, M. A. Salle, *Darboux Transformations and Solitons*, Springer Series in Nonlinear Dynamics 4, Springer-Verlag, Berlin, 1991.

[2] C. Rogers, W. K. Schief, "Bäcklund and Darboux transformations", Geometry and modern applications in soliton theory, *Cambridge Texts in Applied Mathematics*, 2002.

[3] A. I. Bobenko, Yu. B. Suris, "Integrable systems on quad-graphs", *Int. Math. Res. Notices*, **11**, 573–611.

[4] F. Khanizadeh, A. V. Mikhailov, Jing Ping Wang, "Darboux transformations and recursion operators for differential-difference equations", *Theoretical and Mathematical Physics*, **177(3)** (2013), 1606–1654.

[5] A. V. Mikhailov, G. Papamikos, Jing Ping Wang, "Darboux transformation with dihedral reduction group", *Journal of Mathematical Physics*, **55(11)** (2014), 113507, arXiv: 1402.5660.

[6] W. R. Wasow, *Asymptotic expansions of solutions of ordinary differential equations*, Pure and applied mathematics, Wiley Interscience Publishes, New York, 1965.

[7] V. G. Drinfel'd, V. V. Sokolov, "Lie algebras and equations of Korteweg– de Vries type", Itogi Nauki i Tekhniki, **24**, Akad. Nauk SSSR Vsesoyuz. Inst. Nauchn. i Tekhn. Inform., Moscow, 1984, 81–180.

[8] A. V. Mikhailov, A. B. Shabat, "Conditions for integrability of systems of two equations of the form $u_t = A(u)u_{xx} + F(u, u_x)$. I", *Teoret. Mat. Fiz.*, **62(2)** (1985), 163–185.

[9] A. V. Mikhailov, "Formal diagonalisation of Darboux transformation and conservation laws of integrable PDEs, PDΔEs and PΔEs", *International Workshop "Geometric Structures in Integrable Systems"* (October 30 November 02, 2012, M.V. Lomonosov Moscow State University, Moscow), http://www.mathnet.ru/php/presentation.phtml?option_lang=eng&presentid=5934.

[10] A. V. Mikhailov, "Formal diagonalisation of the Lax-Darboux scheme and conservation laws of integrable partial differential, differential-difference and partial difference", *DIS A follow-up meeting* (8–12 July 2013, Isaac Newton Institute for Mathematical Sciences), http://www.newton.ac.uk/programmes/DIS/seminars/2013071114001.html.

[11] I. T. Habibullin, M. V. Yangubaeva, "Formal diagonalization of a discrete lax operator and conservation laws and symmetries of dynamical systems", *Theoretical and Mathematical Physics*, **177(3)** (2013), 1655–1679.

[12] R. N. Garifullin, A. V. Mikhailov, R. I. Yamilov, "Discrete equation on a square lattice with a nonstandard structure of generalized symmetries", *Theoretical and Mathematical Physics*, **180(1)** (2014), 765–780.

[13] V. E. Zakharov, A. B. Shabat, "Exact theory of two-dimensional self-focusing and one-dimensional self-modulation of waves in nonlinear media", *Ž. Èksper. Teoret. Fiz.*, **61(1)** (1971), 118–134.

[14] A. V. Mikhailov, A. B. Shabat, V. V. Sokolov, "The symmetry approach to classification of integrable equations", Springer Ser. Nonlinear Dynamics, Springer, Berlin, 1991, 115–184.

[15] A. V. Mikhailov, editor, "Integrability", *Lecture Notes in Physics*, **767** (2009).

[16] V. E. Adler, *Classification of discrete integrable equations*, DSci Thesis, L. D. Landau Institute, 2010.

[17] I. Merola, O. Ragnisco, Gui-Zhang Tu, "A novel hierarchy of integrable lattices", *Inverse Problems*, **10(6)** (1994), 1315–1334.

[18] L. A. Takhtadzhyan, V. E. Zakharov, "Equivalence of the nonlinear Schrödinger equation and the equation of a Heisenberg ferromagnet", *Theoretical and Mathematical Physics*, **38(1)** (1979), 26–35.

[19] V. E. Zakharov, A. V. Mikhailov, "Relativistically invariant two-dimensional models of field theory which are integrable by means of the inverse scattering problem method", *Zh. Èksper. Teoret. Fiz.*, **74(6)** (1978), 1953–1973.

[20] A. V. Zhiber, V. V. Sokolov, "Exactly integrable hyperbolic equations of Liouville type", *Uspekhi Mat. Nauk*, **56(1(337))** (2001), 63–106.

DOI: 10.18255/1818-1015-2015-6-795-817

Формальная диагонализация схем Лакса–Дарбу

Михайлов А.В.

получена 28 ноября 2015

В статье мы обсуждаем концепцию схемы Лакса–Дарбу и иллюстрируем ее на хорошо известных примерах, ассоциированных с нелинейным уравнением Шрёдингера. Мы изучаем связи, возникшие благодаря преобразованиям Дарбу, между иерархиями нелинейного уравнения Шрёдингера, модели Гейзенберга, модели главного кирального поля, а также дифференциально-разностными системами (такими как цепочка Тоды и дифференциально-разностная цепочка Гейзенберга) и конечно-разностными интегрируемыми системами. Мы показываем, что существует формальное преобразование, которое одновременно диагонализует все элементы схемы Лакса–Дарбу. Это приводит нас к производящим функциям локальных законов сохранения для всех интегрируемых систем, полученных в рамках данной схемы Лакса–Дарбу. Обсуждаются связи между законами сохранения систем, принадлежащих заданной схеме Лакса–Дарбу.

Ключевые слова: формальная диагонализация, схемы Лакса–Дарбу, нелинейное уравнение Шрёдингера

Для цитирования: Михайлов А.В., "Формальная диагонализация схем Лакса–Дарбу", *Моделирование и анализ информационных систем*, **22**:6 (2015), 795–817.

Об авторах:
Михайлов Александр Васильевич – доктор физико-математических наук, профессор,
Школа математики Университета Лидса (Лидс, Великобритания)
University of Leeds, Leeds, LS2 9JT, UK
e-mail: A.V.Mikhailov @ leeds.ac.uk

Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 818–833
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 818–833

DOI: 10.18255/1818-1015-2015-6-818-833

UDC 519.682.6+004.43

System Runs Analysis with Process Mining

Shershakov S. A.[1], Rubin V. A.

Received December 15, 2015

Information systems (IS) produce numerous traces and logs at runtime. In the context of SOA-based (service-oriented architecture) IS, these logs contain details about sequences of process and service calls. Modern application monitoring and error tracking tools provide only rather straightforward log search and filtering functionality. However, “clever” analysis of the logs is highly useful, since it can provide valuable insights into the system architecture, interaction of business domains and services. Here we took runs event logs (trace data) of a big booking system and discovered architectural guidelines violations and common anti-patterns. We applied mature process mining techniques for discovery and analysis of these logs. The aims of process mining are to discover, analyze, and improve processes on the basis of IS behavior recorded as event logs. In several specific examples, we show successful applications of process mining to system runtime analysis and motivate further research in this area.

The article is published in the authors' wording.

Keywords: process mining, software process, software runtime analysis

For citation: Shershakov S. A., Rubin V. A., "System Runs Analysis with Process Mining", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 818–833.

On the authors:
Shershakov Sergey Anreevich, orcid.org/0000-0001-8173-5970, research fellow,
National Research University Higher School of Economics,
20 Myasnitskaya str., Moscow, 101000, Russia,
e-mail: sshershakov@hse.ru

Rubin Vladimir Aleksandrovich, orcid.org/0000-0001-8176-2426, PhD, CEO,
Dr. Rubin IT Consulting,
60599, Frankfurt am Main, Germany,
e-mail: vroubine@gmail.com

Acknowledgments:
[1]This work is supported by the Basic Research Program of the National Research University Higher School of Economics.

Introduction

Processes are all around us. Processes accompany data and are accompanied by data. As processes becomes more complex, the information systems accompanying them become more complex too. Thus, the complexity of modern software systems containing millions of lines of code and thousands dependencies among components is extremely high. Supporting such systems requires involving new techniques and tools responding to the challenge of scale and complexity of modern information systems.

Almost all modern software systems trace data at runtime. Information about failures and exceptions is always traced, but also particular data about system execution, system state, called services and so on. In most cases, traces are the only possibility to understand the behavior of a productive system, which usually runs in a separate production environment and can not be debugged.

Process mining is a discipline, basic research and practical purpose of which is to extract process models from data of a special type, that is *event logs* [1]. The traditional areas of the process mining application include business processes (management), social processes, such as medicine or management of municipalities, technological processes. The Process Mining Manifesto released by the IEEE Task Force on Process Mining [2] in 2011 is supported by more than 50 organizations, more than 70 experts contributed to it. One particularly interesting research area is Software Process Mining (SPM), that deals with extracting models of processes related to design, development, debugging and support of software from event logs containing data that software systems trace at runtime [3, 4, 5].

Speaking of ISs, one can distinguish a separate class of component-based information systems, the main feature of which is the structure in which the expansion of functionality of the IS is achieved by adding special components. One of the most growing and rather young approaches to component design is Service-oriented architecture (SOA) [6]. For such systems, logged data can represent traces of interconnection of their components such as *processes* and *services*.

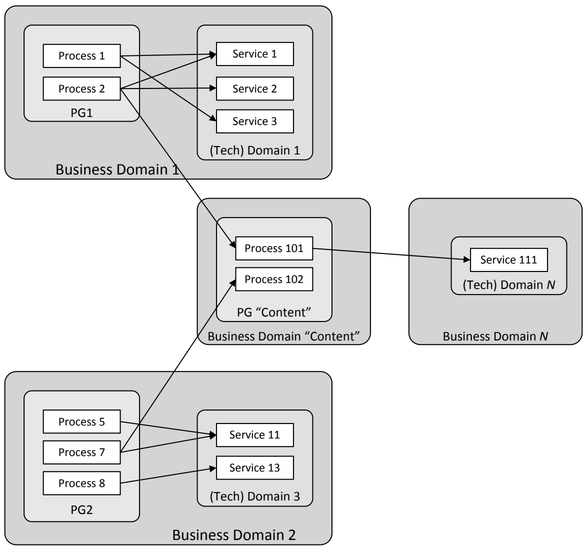


Fig 1. "Domains-Processes-Services" architecture scheme

A large European touristic Computer Reservation System (CRS) will be considered as an example of process-driven system based on SOA architecture. A CRS system we are studying is a distributed SOA-based software containing different client (thin and rich clients) and server components. The system contains *processes* as a basic task of the underlying business logic. Each instance of these processes can be considered as a start point for yet another use case studied with using process mining technics. Processes orchestate *services.* There are a lot of services that can be considered as basic working elements of the system.

There are several *business domains.* Each of them consists of processes and technical services. Each process and service belongs to only one *business domain* according to their business purposes. For example, there are *"booking"* and *"accounting"* business domains. The processes are combined into *process groups* (PG) and the services are grouped into *technical domains* (TD)[1] (Fig. 1). Coming back to the earlier example, there are PG and TD named "booking". Technically all the processes and services are implemented as Java interfaces (correspond to *InterfaceName*) organized as Java-packages (correspond to *PackageName*). Each process and service send and receive messages (correspond to *OperationName*), and each specific message is logged as an individual trace to a log.

We have a set of logs tracing interconnections of those processes and services and containing a lot of different aspects of data. By examining certain *views* of the data we can look at the system from different perspectives. Each specific view can represent some aspect of the system. We distinguish such aspects as *Control Flow Aspect*, *Data Perspective* (also can be treated as *Informational Aspect*), *Organizational Aspect* and *Infrastructural context.* Here we are primarily focusing on the control flow aspect, hovewer at subsequent steps the data aspect is also considered.

Architecture teams normally define a set of rules about how individual processes can invoke other (sub)processes and services from different domains. As an example there are a lot of restrictions about invoking services or processes from other services. These rules and restrictions are also known as *Architectural Guidelines or Architectural Conventions.* Such rules can be complicated enough in order to be simply tested at compile-time, or during the module testing or at runtime. This paper deals with detecting some architectural violations in the model discovered from the event logs derived from a running system.

In the area of software engineering there are a set of well-known *architectural principles*, such as loose coupling, separation of concerns, etc. Our goal here is not only to detect the violations of architectural conventions of the company, but also of the violations of these common architectural principles. We consider the ability to make some kinds of models of a given software system by using process mining approaches.

The rest of this paper is organized as follows. Section 1 presents data logs and tools used for logs analysis. It also contains three examples of violations of architectural conventions and principles detected by using process mining techniques. Section 2 discusses some related work and section 3 summarizes the work done and discusses future work.

[1]We especially refer to them as *technical domains* in order not to mix them with the *business domains.*

1. Experience report

1.1. Log and Tools

An *event log* is the starting point for almost any process mining research.

The subsystems of the CRM system maintains an ability for tracing of all the necessary processes and services communication. The initial invocation of any process, e.g. made by a rich-client application, is accompanied by allocation of a special *invocation id* (also reffered to as `InvID`).

Both processes (PR) and services (SV) receive a request message (RQ) as input and a return response message (RS) or an exception (SE) as output. These messages are logged. On the log level the traces are written in an XML format. A sample of such XML log is presented in the Listing 1. Each trace is included into a `tracingevent` element containing several sections that describe the trace and contain additional data that could be used for deeper analysis. For the first step we are interested in the following data: an *invocation id*, a *message recipient* (given by its full-qualified name including *business domain* (we also refer to it as `UnitName`), *package name*, *interface name*, and operation name).

Then, we also consider two very important fields. An *event timestamp* is first. Describing the second one we have to mention that a trace event contains a *payload*, given in the form used by processes and services to exchange data between each other. Payloads are presented in the form of an XML-based piece of data and can be used to analyze a model made with this log from a data perspective. During this work we are considering only two attributes of the payload: size of its data and its hash sum to identify whether the payload is changed from call to call.

Listing 1. Event trace for resolveLocationByAlias service call

```
<tracingevent>
<InvocationIdentifier>
<id>639041439044799821</id>
<time>Fri Dec 20 00:11:48 CET 2013</time>
...
</InvocationIdentifier>
<TransactionContext>
...
</TransactionContext>
<log4j:event logger="tracer.de.der.pu.domains.geo.
location.LocationQuery.resolveLocationsByAlias"
timestamp="1387494715759" level="INFO"
thread="WorkerThread#8[10.10.10.42:57387]">
<log4j:message><![CDATA[Request]]></log4j:message>
</log4j:event>
<jboss>
...
</jboss>
<payload>
<![CDATA[
<ServiceRequest>
...
</ServiceRequest>
]]>
</payload>
</tracingevent>
```

In this paper we are considering a log produced by the system during a period of 24 hours. The size of all XML files of the log is approximately 10 GB.

1.1.1. Log traces "normalization" with an RDBMS approach

The event log has a complex structure and contains heterogenous data covering different aspects, such as control flow, organizational aspect (including user information, authentication and authorisation info), infrastructural context and other resources. It is necessary to represent them in a formalized form.

We decided to use RDBMS as a log representation because it provides the possibility to make various data views for different types of analysis depending on each specific aspect containing in our log. Also, it is an instrument for effective manipulation of big amounts of log data.

Creating a view on data involves two operations. First, *projection* of a data set (that could be a regular table as well as a joint of a number of tables) performs selecting only a particular subset of all *attributes* (table columns) that correspond to specific aspects. Second, *filtering* of the dataset provides only those records (table rows) which match some selection criteria.

Converting a text-based logs into a well-defined RDBMS allows us to obtain a desirable data projection on a specific aspect in a very natural way just by pointing all the necessary atributes out. At the same time it is very simple to obtain a filtered subset of events by specifying arbitrary filtering criteria [1, 261]), with subsequent export as frequently as it is necessary. Using indexes prepared in a proper way allows performing such operations quickly enough even on a very big amount of data. Similar approach was effectively used in other case studies [7].

We decided to use SQLite database engine to store and manipulate the data. A part of a relation diagram for the log database is depicted on Fig. 2. Here, we distinguish two main parts of physical storage. One part consists of declarations of interfaces including PR/SV type, business domain unit name, package name, and interface name. The part is represented as a table with `Interfaces` name. All other data corresponding to the log's trace events are represented by `TracingEvent` table. For our cases we used joint records with `Interfaces.ID` and `TracingEvents.Interface_ID` fields as keys. We refer to those as `FullData3` view below.

An example of records corresponding to the trace described by Listing 1 is given on Fig. 3.

We developed a tool performing parsing of XML source files and adding parsed data to a database. For the given 10 GB log (discussed in sect. 1.1), it takes at least an hour to convert all the XML data to the DB format using a laptop with an Intel® i3 @ 2,4 GHz comparable processor on the board[2].

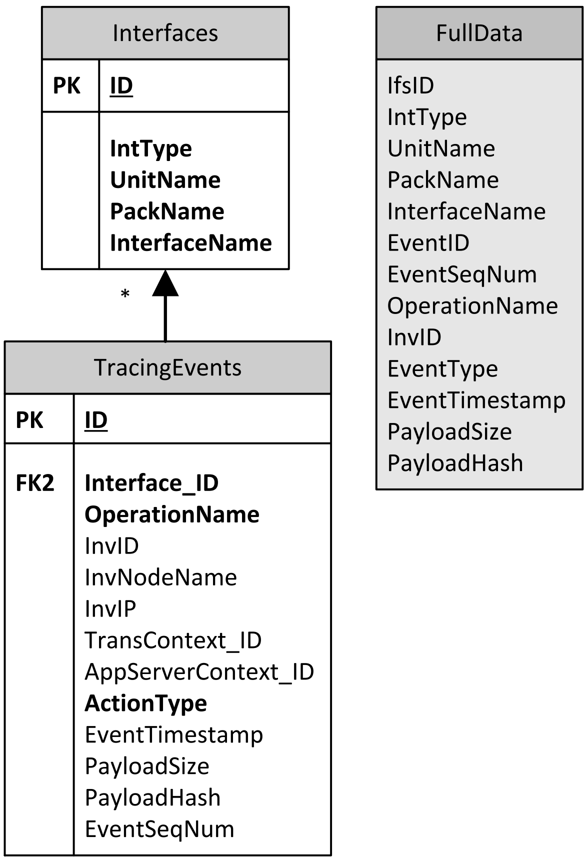


Fig 2. A relation diagram of a log DB

RecNo	ID	PDType	UnitName	PackName	InterfaceName
1	110	SV	geo	location	LocationQuery

RecNo	ID	Interfa-ce_ID	Operation Name	InvID	EventType	Event Timestamp	Payload Size	Event Seq Num
1	170097	110	resolve Locations ByAlias	639041439 044799000	RQ	1387494715759	393	11

Fig 3. DB records for a trace from Listing 1 (table `Interfaces` at top and table `TracingEvents` at bottom)

“Normalizing” the log by converting it to the DB allows making a rather compact representation of source data. Thus, for the 10 GB containing approximately 500000 traces there is just 60 MB of data given in SQLite DB ver.3 format. Moreover, reexporting a full set of (merged) data from the DB to an external CSV-text file takes just a couple of seconds.

As shown further, using SQL queries for extraction of a precisely needed data projection is significantly efficient. Thus, the SQL-based approach is one of the basic tools used in this work.

[2]Among the factors significantly affecting the converting speed, one can distinguish a hashing algorithm.

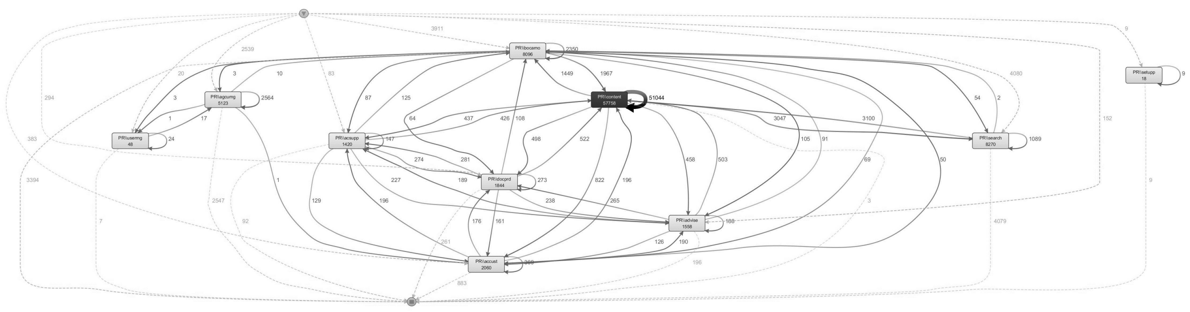


Fig 4. Fuzzy diagram representing relations between processes, $AP(100, 100)$

1.1.2. Process mining specific tools

Today, there is a set of freeware and commercial process mining tools. Examples of these tools include ProM [8], which is a widely used research workbench containing more then 600 plugins. Disco is another well known process mining tool [9].Disco is based on Fuzzy miner which was initially implemeted in ProM [10].

In [11] we show an RDBMS-based approach to automation process mining experiments with rapid creation of datasets using VTMine Framework, which is our own tool for modeling and conducting process mining experiments. DPModel [12, 13], a graphical language for automation of experiments in Process Mining, underlies the tool.

We decided to mine a process model of our system as a *fuzzy model* that can be discovered with Disco. It has a good usability and good performance in processing large event logs. Moreover, it can perform advanced filtering on the basis of Fuzzy miner.

In the further sections, with the help of process mining techniques we analyze the logs in order to find software architectural violations and inconsistencies.

1.2. Example 1: Architectural Violations

In Introduction we have already discussed Architectural Conventions and Architectural Violations. Invoking some processes from other processes can be an example of such violation. In our CRS the only processes from “content” PG (and, respectively, business domain) can be invoked from other processes. Calling any other processes from different domains are not allowed. In this section we investigate the presence of exactly this violation.

We can detect such forbidden calls from different domains by investigating a fuzzy diagram.

The very important assumption we have to make is that there are no asynchronous messages calls between processes and services in the scope of the set of events related to one specific case.

1.2.1. Model in Disco

Now we are looking at Control Flow Aspect for discovering forbidden services calls. In order to discover process calls violations we build a Disco model depicting PG/Domain relationships.

Although all event attributes can be used for process mining, we focus on the two attributes that are mandatory for process mining. Any event should refer to a *case* (i.e., a process instance) and an *activity*. Moreover, events related to a particular case should be ordered. Thus, for performing data import to Disco from a RDBMS one need to point out which attributes are used for indicating *case*, *activity* and *timestamp* (for ordering reasons). Other attributes are indicated for Disco as *resources* that can be used for filtering.

In this work, we are using *invocation identifier* given as `InvID` attribute as a *case* ID and `EventTimestamp` attribute as a *timestamp*. Choosing attributes that play a role of *activity ID* is closely related to the studying case. For this case, we use a pair of attributes `IntType` (which can be either `PR` or `SV` for processes/PG and services/domains respectively) + `UnitName` (representing the name of PG or domain and corresponding to business domain) as an *activity ID*.

There are two parameters of Disco's Fuzzy Miner: (1) number of activities, and (2) number of paths shown on a fuzzy map. They are used to make a quick filtration by the criteria of frequency of activities and paths being met while making the map. We refer to both of these parameters as a pair, e.g. $AP(100, 100)$, where the numbers in the brackets are the percentage of activities and paths, respectively.

There is a fuzzy map with $AP(100, 100)$ produced by Disco depicted on Fig. 4. This map is a graph that demonstrates the relations between both PGs and domains transparently given through the messages sent by their interfaces. The map was built from a filtered dataset in order to restrict the model to view only **processes**. For this very purpose we can apply Disco *attribute filter* selecting only the activities containing `PR\\` we would like to observe.

The map contains only 10 vertices and a number of arcs which is not so big so we are able to track individual relations between each pair of processes. Vertices color coding shows us how many messages are sent to individual vertices. Thus, `PR\\content` is represented as a dark blue vertex showing us that it has many more incoming messages (57758) than the others. This is because process `content` plays a special role in the whole system (as it was mentioned above). In other words, it contains common "routines".

At the same time we can remark the presence of pairs of income/outcome edges between other PGs as well as those which contain only few traces. The direct communication of the PGs with each other represent examples of violations of the architectural conventions.

To investigate this violation more precisely we will first make a more detailed filtration by using Disco. As a concrete example one can consider inappropriate direct relations between two PGs — `bocamo` and `search`. We use the so-called *Follower Filter* that lets us define a couple of activities with a restriction of how they should follow one another. Thus we set a filter a way that `search` activity follows `bocamo` activity (Fig. 5).

We can see that there are 54 messages from `bocamo` to `search`. Here, we understand a message as a pair of events, first of which corresponds to `PR\\bocamo` activity and is directly followed by second one corresponding to `PR\\search` activity. Such pairs are found in a number of cases, and some of the cases contain such pairs several times. Depending on event activities and their order in the cases the latter are grouped into so-called *variants*.

By using Disco case statistics we can see there are 10 different variants of traces

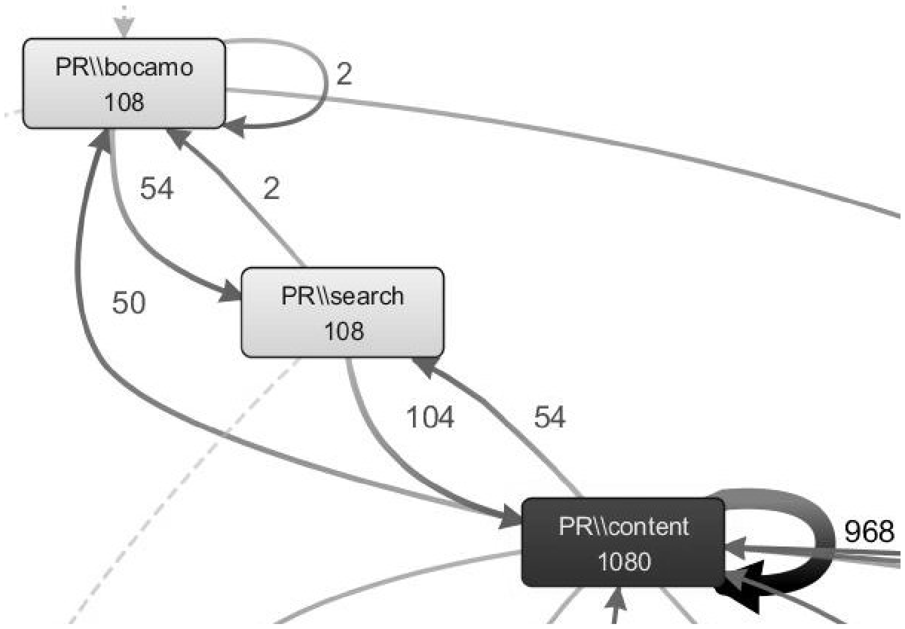


Fig 5. Correlation between activities `search`, `bocamo`, and `content`

including from 1 to 40 cases per variant. For example, variant 2 contains 5 cases, from which we have detected one (with invocation id 190248972438317148) when bocamo process `quoteItemFromSearch` calls search process `getPricesForRouting` and it is forbidden. See a sequence diagram in Fig. 7.

PG	Operation	Type
bocamo	quoteItemFromSearch	RQ
search	getPricesForRouting	RQ
content	priceAvailability	RQ
content	priceAvailability	RS
content	calculateFees	RQ
content	calculateFees	RS
search	getPricesForRouting	RS
content	determineBestProduct	RQ
content	determineBestProduct	RS
content	checkBookingUnitRestriction	RQ
content	checkBookingUnitRestriction	RS
content	getExternalReferences	RQ
content	getExternalReferences	RS
content	performQuote	RQ
content	performQuote	SE
bocamo	quoteItemFromSearch	SE

Fig 6. Subset of events of case 190248972438317148 related to PGs `bocamo`, `search` and `content`

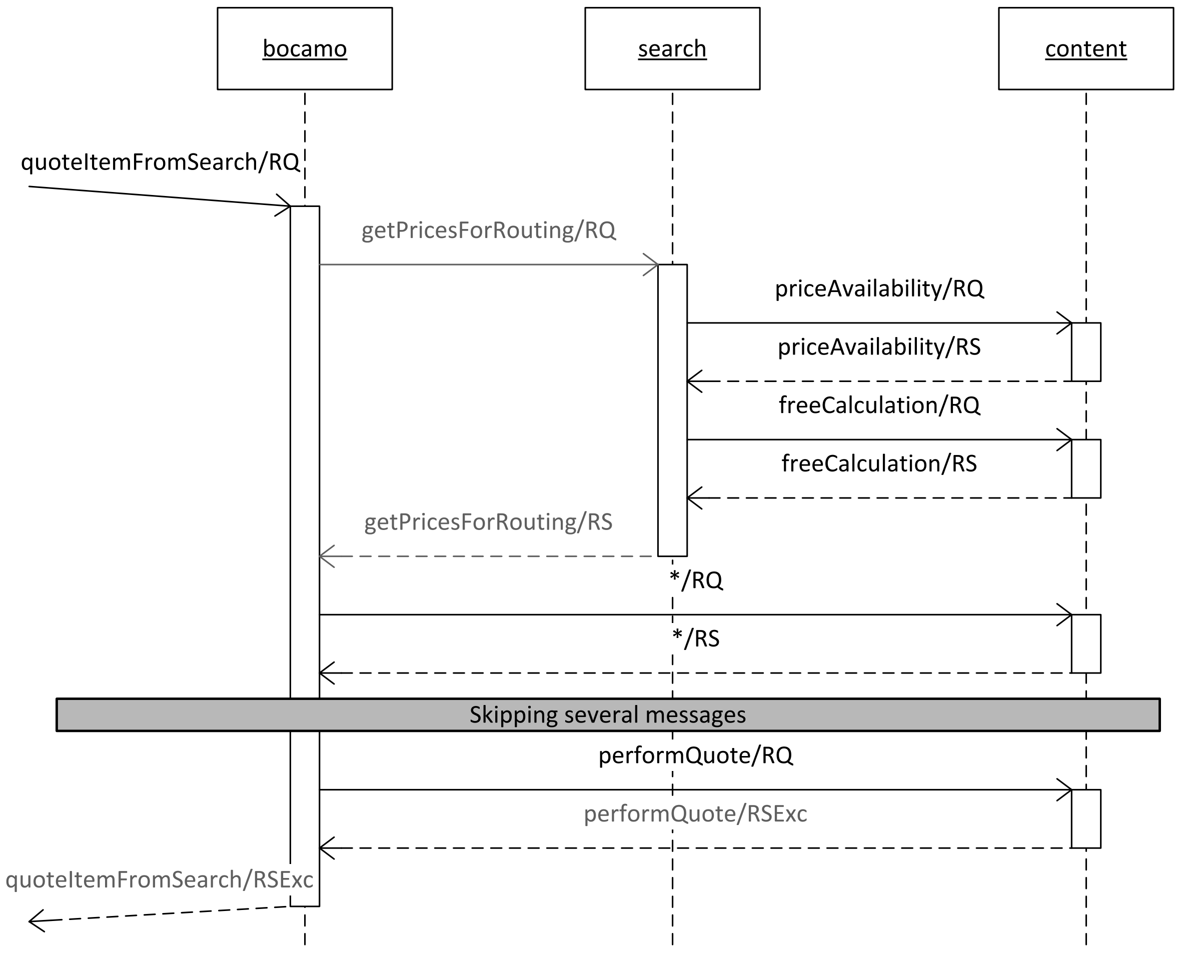


Fig 7. UML diagram corresponding to case 190248972438317148

On Fig. 7, a manually created UML sequence diagram based on the data of the 16-event subset above is depicted. It can be used as a convenient tool for reporting detected violations to developers.

1.2.2. Generalization in SQL

In order to obtain a collection of all the similar violations we created a set of SQL queries. These queries are based on the idea of a square Cartesian product of the set of all the traces. As a result of such product and futher filtering by the same case ID for both parts of the product, we obtained a collection of so-called N-step relations between all the events. Each N-metric is calculated as a difference between relative time positions of each events in one given case. Among the pairs from this relation there are some pairs that contain 1-step relations. Every such relation corresponds to a couple of events among which the first event is directly followed by the second one. Finally, we have to filter the resulting dataset by setting desirable restrictions, such as:

1. both parts of a pair contain different PGs;
2. no part of a pair contains `content` PG;

3. both parts of a pair contain RQ messages, so we are interested only in "request → request" processes with different PGs that is actually forbidden;

As a result of excecuting the SQL query we obtain a set of 2031 traces that produce system architectural violations. The set is mapped to a set of cases containing these events. By constructing one more query grouping cases by `InvID` attribute, we obtain a set of cases containing at least one violation. So, we have found exactly 1789 cases that violate discussed architectural violations. The maximum number of forbidden 1-step relations per case is 3 (for 34 cases). Typical number is just 1 per a case (1581 cases).

1.2.3. Benefits

We have found the violation and reported an appropriate bug to the developers with a view to creating a ticket in a bug-tracking system. Using a notion of UML sequence diagrams may help showing to developers an explicit fragment of the system where the violations are detected.

1.3. Example 2: Antipattern "Unnecessary repeating calls"

Let us now look for more oddities by applying the same techniques.

Thus, we set `OperatinName` in conjuction with `IfsID` and `EventType` attributes as activity. Looking at the statistics panel in Disco one can conclude that among the most often used activities there is `getAgency` activity in `agcumg` domain: there are 7642 RQs and RSs instances. Let us consider it more precisely. Applying an attribute filter for this activity name and marking it as "mandatory" we obtain a fuzzy map which, depending on $AP()$ value, contains a small or large number of activities, but under the condition that all these activities are related to `getAgency`. Thus, there is a number of cases containing `getAgency` as a repetitive activity. Among the latter, `getInvoiceListByReservationNumber` in `accust` PG can be identified.

Such cases are characterized by the fact that they have repetitive messages `getInvoice ListByReservationNumber` ↔ `getAgency` with the same payload (given by its size and hash sum), that can lead to the fact of presence of multiple excessive process invocations. This can signify to us a **bad implementation example** where payload data must be cached or stored instead of their repititive obtaining.

Let us consider a technique for detecting such patterns. The first step is creating of an auxilary view `ActivitiesCountPerCases1` that for each case contains a set of activities (identified separately for RQ/RS) with a number of their repetitions:

```
CREATE VIEW [ActivitiesCountPerCases1] AS
SELECT *, COUNT(*) AS ActsNum
FROM FullData3
GROUP BY InvID, IfsID, OperationName, EventType;
```

Then we can fix a certain threshold number of activities' repetitions and select only the instances of repetitive activities (make another auxilary view `Activities3PerCase`):

```
CREATE VIEW [Activities3PerCase] AS
SELECT InvID, IfsID, OperationName
```

```
FROM ActivitiesCountPerCases1
WHERE ActsNum >= 3;
```

Next step: retrieving full attributed data for all the activities satisfying the condition above (another view `EventsBy3ActivitiesPerCase1`):

```
CREATE VIEW [EventsBy3ActivitiesPerCase1] AS
SELECT *
FROM Activities3PerCase AS L INNER JOIN FullData3 AS R
ON (L.InvID = R.InvID AND L.IfsID = R.IfsID AND
L.OperationName = R.OperationName)
GROUP BY ID;
```

Finally, we form a resulting set according to identical values of `PayloadSize` and `PayloadCache`:

```
SELECT *, COUNT(*) AS NumOfRepeatedPayload
FROM EventsBy3ActivitiesPerCase1
GROUP BY InvID, IfsID, OperationName, EventType,
PayloadSize, PayloadCache
```

By filtering `NumOfRepeatedPayload` attribute by number of maximum allowable repetitions we obtain all the events (and consequently cases) with excessive payload transmitting.

1.4. Example 3: Antipattern “Cross-cutting concern”

According to the statistics, `getConfiguration` in `smerge` domain is the most often invoked service (aprx. 12 % of all traces). Let us observe how `smerge` domain (both as business and techical means) is related to other PGs.

First, we set a pair of attributes `IntType` and `UnitName` as activity. In order to eliminate irrelevant cases we add some filters: (1) selecting only RQ traces, (2) selecting only processes from all domains and services in `smerge` domain, (3) marking `getConfiguration` operation name as mandatory. The resulting fuzzy map is depicted on Fig. 8.

As we can conclude, `smerge` domain is actively “invoked” by 7 other PGs. Precisely, operation `getConfiguration` is invoked by the processes contained in these PGs. So, this operation can be considered as a so-called “Cross-cutting concern” and must belong to a dedicated domain (but not to `smerge` domain) or should be moved to `context` domain.

2. Releated work

There are various works on *software execution traces* and *runtime analysis*. In [14], two runtime analysis algorithms, a data race detection algorithm and a deadlock detection algorithm, are introduced to analyze Java programs. The concerned approach is based on the idea of *single* program execution and observing the generated run to extract various kinds of information. In contrast with this approach, process mining works with a set of traces, but not with only one trace, despite using algorithms.

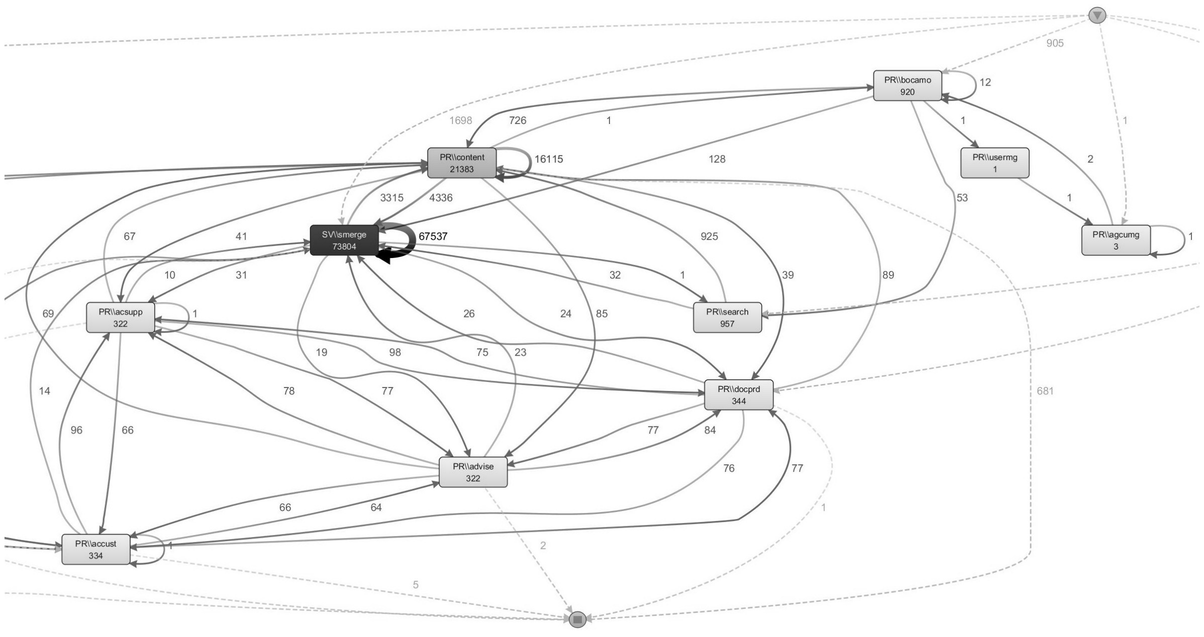


Fig 8. Fuzzy diagram representing relations between PG and a `smerge` domain, $AP(100, 100)$

An approach to recover interaction patterns between different entities such as methods, files, or modules, based on analysis and comparision of execution traces of different versions of a software system, is proposed in [15]. One of the goals is to track the evolution of particular modules and to visualize the findings. Like in our paper, the authors use a standard database technology for maintaining analyzed data.

Both of the approaches above and a number of other "classical" approaches to runtime analysis is based on an idea of code instrumenting [16]. Process mining does not require any specific code instrumenting and utilize any traces that software can produce during its execution.

A *trace summarization* technique for manipulating traces, based on metrics for measuring various properties of an execution trace, was introduced in [17]. It is proposed to use trace summaries to enable top-down analysis of traces as well as recovery of system behavioural models. There was proposed a trace summarization algorithm that is based on successive filtering of implementation details from traces.

An idea to apply process mining to services, so-called *service mining*, was proposed in [18, 19]. Finally, based on the process mining discipline, a comprehensive approach to diagnostic information in compliance checking was proposed in [20]. We suppose that using a similar approach for Petri net models of the processes discussed in our paper can introduce some new ideas for achieving the objectives of SRA.

One of the most novel approaches to reverse engineering for obtaining real-life event logs from distributed systems is presented in [21]. The approach allows to analyze operational processes of software systems under real-life conditions and use process mining techniques to obtain precise and formal models.

3. Future work and conclusion

The results obtained during the first practical experiments show us several ways for future work.

First, taking into account specificity of the subject domain, which is software architecture and engineering, introduction of convenient and accustomed tools particularly for model representation is desirable. As an example, the UML sequence diagrams miner mentioned in sec. 1.2.1 can be considered, possibly based on ProM or any other tool. Now we have to construct a UML sequence diagram manually, and it would be a good challenge to provide ability for constructing such diagrams automatically, e.g. by a special ProM plug-in.

Then, it is rather desirable to obtain other kinds of models to provide more comprehensive analysis by using different mining algorithms. Nevertheless, it is still a problem to process large logs with a full range of academic tool basically implemented as ProM plug-ins. Also, we suppose that it is possible to use these models to detect/recognize other architectural patterns that can be used for improving systems in some ways. In order to do this one needs to consider also other methods for pattern recognition in a model like the one proposed in *compliance checking* research [20].

In this paper only few violations of architectural principles and patterns are concerned. One of our goals is to create a catalog of architectural patterns/architectural violations related to different kinds of systems.

Finally, there is also *Software Performance Analysis* which is a separate big problem we would like to investigate with the help of process mining techiques.

4. Acknowledgment

The authors would like to thank Fluxicon for powerfull process mining tool Disco provided.

References

[1] W. M. P. van der Aalst, *Process Mining – Discovery, Conformance and Enhancement of Business Processes*, Springer, 2011.

[2] IEEE Task Force on Process Mining, "Process Mining Manifesto", *BPM 2011 Workshops, ser. Lecture Notes in Business Information Processing*, **99**, eds. F. Daniel, S. Dustdar, K. Barkaoui, Springer-Verlag, Berlin, 2011, 169–194.

[3] E. Kindler, V. Rubin, W. Schäfer, "Activity mining for discovering software process models", *Software Engineering*, **79**, eds. B. Biel, M. Book, V. Gruhn, 2006, 175–180.

[4] V. Rubin, I. Lomazova, W. M. van der Aalst, "Agile development with software process mining", *ICSSP 2014*, ACM, Nanjing Jiangsu, China, 2014, 70–74.

[5] V. Rubin, A. A. Mitsyuk, I. A. Lomazova, W. M. P. van der Aalst, "Process mining can be applied to software too!", *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, NY: ACM, 2014.

[6] J. McGovern, O. Sims, A. Jain, M. Little*Enterprise Service Oriented Architectures: Concepts, Challenges, Recommendations*, Springer, 2006.

[7] A. Mitsyuk, A. Kalenkova, S. Shershakov, W. van der Aalst, "Using process mining for the analysis of an e-trade system: A case study", *Software Engineering (in Russian)*, **3**, 2014, 15–27.

[8] H. Verbeek, J. Buijs, B. Dongen, W. Aalst, "ProM 6: The Process Mining Toolkit", *Proc. of BPM Demonstration Track 2010, ser. CEUR Workshop Proceedings*, **615**, eds. M. L. Rosa, 2010, 34–39.

[9] [Online]. Available: http://www.fluxicon.com/disco.

[10] C. W. Günther, W. M. P. Van Der Aalst, "Fuzzy mining: Adaptive process simplification based on multi-perspective metrics", *Proceedings of the 5th International Conference on Business Process Management, ser. BPM'07*, Springer-Verlag, Berlin, Heidelberg, 2007, 328–343.

[11] S. A. Shershakov, "VTMine framework as applied to process mining modeling", *International Journal of Computer and Communication Engineering*, **4**:3 (2015), 166–179.

[12] S. Shershakov, "DPMine/P: modeling and process mining language and ProM plug-ins", *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, eds. A. N. Terekhov, M. Tsepkov, ACM New York, NY, USA, 2013.

[13] S. A. Shershakov, "DPMine graphical language for automation of experiments in process mining [in russian]", *Modeling and Analysis of Information Systems*, **21**:5 (2014), 102–115.

[14] K. Havelund, "Using runtime analysis to guide model checking of java programs", *SPIN*, Lecture Notes in Computer Science, **1885**, eds. K. Havelund, J. Penix, W. Visser, Springer, 2000, 245–264.

[15] M. Fischer, J. Oberleitner, H. Gall, T. Gschwind, "System evolution tracking through execution trace analysis", *IWPC*, IEEE Computer Society, 2005, 237–246.

[16] T. Ball, "The concept of dynamic analysis", *ESEC / SIGSOFT FSE*, Lecture Notes in Computer Science, **1687**, eds. O. Nierstrasz, M. Lemoine, Springer, 1999, 216–234.

[17] A. Hamou-Lhadj, *Techniques to simplify the analysis of execution traces for program comprehension*, Ph.D. dissertation, Ottawa-Carleton Institute for Computer Science School of Information Technology and Engineering, University of Ottawa, 2005.

[18] W. Aalst, H. Verbeek, "Process Mining in Web Services: The WebSphere Case", *IEEE Bulletin of the Technical Committee on Data Engineering*, **31**:3 (2008), 45–48.

[19] W. van der Aalst, "Service mining: Using process mining to discover, check, and improve service behavior", *IEEE Transactions on Services Computing*, **99**:PrePrints (2012), 1.

[20] E. Ramezani, D. Fahland, B. F. van Dongen, W. M. P. van der Aalst, *Diagnostic information for compliance checking of temporal compliance requirements*, Tech. Rep., 2013., [Online]. Available: http://dblp.uni-trier.de/db/conf/caise/caise2013.html#TaghiabadiFDA13.

[21] M. Leemans, W. M. P. van der Aalst, "Process mining in software systems: Discovering real-life business transactions and process models from distributed systems", *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*, 2015, 44–53.

DOI: 10.18255/1818-1015-2015-6-818-833

Анализ системных исполнений с помощью Process Mining

Шершаков С. А.[1], Рубин В. А.

получена 15 декабря 2015

Информационные системы (ИС) оставляют многочисленные следы и журналы событий своей работы. В контексте сервисно-ориентированной архитектуры (СОА) информационной системы такие журналы содержат детальную информацию о последовательностях вызовов процессов и сервисов. Современные инструменты мониторинга приложений и отслеживания ошибок их исполнения предоставляют довольно простые средства поиска и фильтрации журналов событий. Тем не менее, "интеллектуальный" анализ таких журналов событий является крайне полезным, так как может предоставить ценную информацию об архитектуре системы, взаимодействии между бизнес-доменами и сервисами. В работе рассматриваются журналы событий (представляющие данные о системных исполнениях) большой информационной системы поддержки бронирования, на основании данных которых производится обнаружение нарушений архитектурных принципов взаимодействия компонентов и общих антипаттернов СОА. Для анализа этих журналов применяются проверенные подходы дисциплины извлечения и анализа процессов (process mining). Process mining применяется для автоматического синтеза моделей процессов, анализа этих процессов и их улучшения на основе информации о поведении ИС, записанной в виде журналов событий. На базе нескольких конкретных примеров демонстрируется успешное применения подходов process mining для анализа системных исполнений и приводится обоснование необходимости дальнейших исследований в данной области.

Статья публикуется в авторской редакции.

Ключевые слова: извлечение и анализ процессов, программные процессы, анализ системных исполнений

Для цитирования: Шершаков С. А., Рубин В. А., "Анализ системных исполнений с помощью Process Mining", *Моделирование и анализ информационных систем*, **22**:6 (2015), 818–833.

Об авторах:
Шершаков Сергей Андреевич, orcid.org/0000-0001-8173-5970, научный сотрудник НУЛ ПОИС ФКН,
Национальный исследовательский университет Высшая школа экономики
101000 Россия, г. Москва, ул. Мясницкая, 20,
e-mail: sshershakov@hse.ru

Рубин Владимир Александрович, orcid.org/0000-0001-8176-2426, PhD, CEO
Dr. Rubin IT Consulting,
60599, Frankfurt am Main, Germany,
e-mail: vroubine@gmail.com

Благодарности:
[1]Работа выполнена в рамках Программы фундаментальных исследований НИУ ВШЭ в 2015 году.

Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 834–851
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 834–851

DOI: 10.18255/1818-1015-2015-6-834-851

УДК 004.853

Лингвостатистический анализ терминологии для построения тезауруса предметной области

Каряева М. С.[1]

получена 15 мая 2015

Работа посвящена анализу корпуса терминов и терминологических источников с целью дальнейшей автоматизации построения тезауруса данной предметной области, в качестве которой рассматривается поэтология. Предварительная систематизация терминологии с использованием лингвостатистического подхода формирует корпус семантически связанных понятий для автоматизации извлечения семантических отношений между терминами, определяющих структуру тезауруса указанной предметной области.

Ключевые слова: тезаурус, метрики семантической близости, data mining, компьютерная лингвистика

Для цитирования: Каряева М. С., "Лингвостатистический анализ терминологии для построения тезауруса предметной области", *Моделирование и анализ информационных систем*, **22**:6 (2015), 834–851.

Об авторах:
Каряева Мария Сергеевна, orcid.org/0000-0003-4466-1735, аспирант,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия,
e-mail: mari.karyaeva@gmail.com

Благодарности:
[1]Работа поддержана грантом РФФИ № 13-06-00448

Введение

В последнее время возрос интерес к такому виду представления знаний, как тезаурус. Под тезаурусом обычно понимают словарь концептов с определенной структурой хранения данных и набором семантических отношений, указывающих на общность (например, синонимический ряд) или противопоставление значений лексических единиц. Главное отличие тезауруса от словаря – это система представления данных, которая позволяет использовать тезаурус не только как средство для отображения информации в удобочитаемом виде, но и для дальнейшей работы с ним, как с источником знаний для задач, связанных с компьютерной лингвистикой и информационным поиском. Например, российский тезаурус РуТез [1] используется в качестве ресурса для автоматической обработки текстов в проектах для государственных и коммерческих организаций. РуТез относится к классу тезаурусов типа WordNet [2], то есть базы знаний знаменательных частей речи (существительных, глаголов, наречий, прилагательных), где лексической единицей является «синсет», или синонимический ряд, то есть набор слов со схожим значением. Тем самым, за счет семантической сети между концептами возможен автоматический анализ текстов и ряд других задач, которые возможно решить с помощью такого мощного инструмента.

Наличие предметно-ориентированного тезауруса позволяет значительно упростить процесс сбора, формализации, хранения, оценки и использования знаний, что способствует повышению эффективности работы специалиста или рабочей группы выбранной предметной области.

Создание тезаурусов предметных областей – достаточно трудоемкий и дорогостоящий процесс, поскольку при этом необходимо объединение усилий целых групп соответствующих специалистов и экспертов для обработки большого числа объемных источников информации: словарей, справочников, научных публикаций и других текстов. Естественным подходом к созданию тезауруса в связи с развитием информационных технологий является комбинирование как ручных, так и автоматических методов.

В данном случае в качестве предметной области выбрана поэтология, под которой понимается группа дисциплин, ориентированных на всестороннее теоретическое и историческое изучение поэзии. Для тезаврирования поэтологии были заложены достаточные основы [3, 4], в которых экспертами предметной области был разработан базовый терминологический словник тезауруса, насчитывающий полторы тысячи специальных терминов, представленных в 10 предметных подобластях (кластерах).

Представлялось логичным для ускорения заполнения основных информационных полей терминологических статей тезауруса (ТСТ) создавать тезаурус как открытый сетевой ресурс с полноценным доступом пользователей к составлению и редактированию корпуса ТСТ [5]. Однако после размещения в сети Интернет прототипа тезауруса [6] с использованием Wiki-технологий стало ясно, что на этом этапе такой краудсорсинговый подход к созданию тезауруса не эффективен в связи с недостатком мотивации у пользователей самостоятельно развивать лингвистические ресурсы. Поэтому оказался методологически актуальным подход с автоматическим составлением тезауруса на основе терминологии предметной области, а именно имеющегося базового терминологического словника тезауруса и ряда источников первичной информации, содержащих информацию для заполнения основных полей ТСТ, как явную – для определения термина, так и неявную – для семантических связей данного термина с другими. Автоматизация лингвостатистического анализа этой терминологии с необходимостью предваряет задачу автоматического составления тезауруса.

Изучение основных закономерностей при построении тезауруса поможет формализовать основные шаги и методики для создания баз знаний и способствовать развитию тезаурусов в других предметных областях. В качестве метода для автоматического извлечения семантических отношений был выбран метод, основанный на извлечении информации из определений рассматриваемых терминов. Семантическая близость терминов оценивается с помощью двух метрик близости и ручной оценки эксперта предметной области.

В качестве отношений между терминами для автоматического извлечения могут быть рассмотрены:

1. Родо-видовые отношения;

2. Целое–части;

3. Отношение синонимии.

Автоматическое распознавание семантических отношений возможно с помощью лексико-синтаксических шаблонов [7]. В данной статье были представлены достаточно простые конструкции для извлечения родо-видовых отношений при построении тезауруса WordNet. Извлечение семантических отношений без идентификации по типам представлено в работе [8]. Метод заключается в использовании статей Википедии для реализации методов машинного обучения, основанных на алгоритмах ближайших и взаимных ближайших соседей и двух метриках семантической близости слов. Результаты извлечения используются в системе Serelex [9] для поиска семантически связанных слов.

Для извлечения родо-видовых отношений был достаточно успешно использован подход [10] на основе использования определений толковых словарей с наложением ряда правил для распознавания отношений между определяемым словом в качестве рода и одним из слов дефиниции в качестве вида.

1. Формальная постановка задачи

Пусть $C = \{c_1, c_2, \ldots, c_N\}$ — множество терминов, $D = \{d_1, d_2, \ldots, d_N\}$ — множество определений, R — пары семантически связанных терминов.

Тогда задачей алгоритма служит задача распознавания множества семантических отношений, представленных в виде $R = \{(c_i, c_j), (c_{i+1}, c_{j+1})\}$ из всевозможных пар терминов. Т.е. необходимо построить функцию $F : R \subseteq C \times C \to \{0, 1\}$ и выбрать пары терминов, для которых $F = 1$.

2. Исходные данные

2.1. Базовый терминологический словник

Экспертами предметной области была разработана база тезауруса, состоящая из списка разделов и терминологического словника, который насчитывает 1545 уникальных терминов.

2.2. Источники терминологических данных

В качестве источников для извлечения семантических отношений и автоматического заполнения полей терминологических статей тезауруса оцифрованы и преобразованы в единую машиночитаемую форму представления следующие источники:

1. Краткая литературная энциклопедия: В 9 т. (КЛЭ) [11];
2. Литературная энциклопедия: В 11 т. (ЛЭ) [12];
3. Словарь литературных терминов: В 2-х т. (СЛТ) [13];
4. Квятковский А.П. Поэтический словарь. (ПСК) [14];
5. Большая советская энциклопедия: В 30 т. (БСЭ) [15].

3. Выбор инструментария для анализа данных

Известен целый ряд различных процедур анализа текста на естественном языке:

- графематический анализ с разделением входного текста на слова и разделители и с выделением предложений, абзацев, заголовков и примечаний;
- лексический анализ (токенизация) входного текста с выделением из него лексем (токенов);
- частеречная разметка, задачей которой является определение части речи и грамматических характеристик слов в тексте;
- морфологический анализ с распознаванием грамматической формы слов и словосочетаний и снятием омонимии;
- лемматизация слова (словоформы), т.е. приведение словоизменительной формы слова к лемме — к её нормальной (словарной) форме;
- стемминг — отбрасывание изменяемых частей слов, преимущественно окончаний;
- синтаксический анализ (парсинг) с распознаванием синтаксического строения предложения;
- семантический анализ, с установлением семантических связей (отношений) между элементами текста, которые могут включать более одного слова.

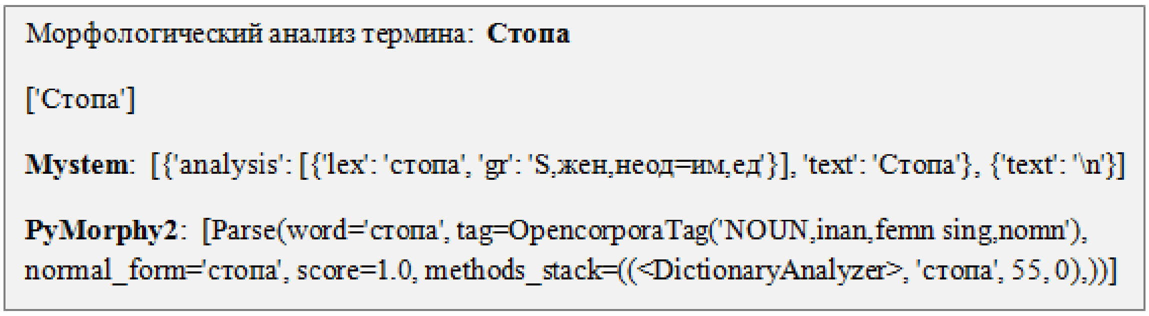
Автоматизация названных процедур для больших объемов данных реализуется множеством способов с помощью разнообразных инструментов и всевозможных анализаторов.

Инструмент автоматической обработки текстов под названием АОТ содержит графематический, морфологический, синтаксический и семантический анализаторы. Морфологический анализатор построен с использованием грамматического словаря А.А. Зализняка [16].

Программа от компании «Яндекс» Mystem [17] позволяет производить морфологический анализ текста на русском языке. Данный анализатор [18] разработан с использованием словаря в виде леса инвертированных префиксных деревьев суффиксов и инвертированного префиксного дерева основ. Mystem позволяет определять начальную форму слова, т.е. производить лемматизацию, определять грамматические характеристики. Дополнительным плюсом данного продукта является возможность построения гипотетических разборов для слов, не входящих в словарь. Для данного инструмента существует wrapper (обертка) PyMystem [19], используемая в качестве библиотеки для языка Python.

Другой анализатор, который возможно интегрировать в Python-проект в качестве библиотеки, называется Pymorphy2 [20], с помощью которого возможно выполнять лемматизацию и анализ слов, осуществлять склонение по заданным грамматическим характеристикам слов. В процессе работы с русским языком используется словарь OpenCorpora [21]. Диапазон скорости разбора более 100 тыс. слов/сек. Кроме того, у Pymorphy2, как и у PyMystem, предусмотрена возможность интеграции с фреймворком Django [22].

Ниже представлен разбор термина «Стопа» средствами Mystem и PyMorphy2.


```
Морфологический анализ термина:  Стопа

['Стопа']

Mystem:  [{'analysis': [{'lex': 'стопа', 'gr': 'S,жен,неод=им,ед'}], 'text': 'Стопа'}, {'text': '\n'}]

PyMorphy2:  [Parse(word='стопа', tag=OpencorporaTag('NOUN,inan,femn sing,nomn'),
normal_form='стопа', score=1.0, methods_stack=((<DictionaryAnalyzer>, 'стопа', 55, 0),))]
```

В качестве инструмента для проведения исследований достаточно использовать высокоуровневый язык программирования Python. Реализация каждого логического шага представлена отдельными скриптами. Полученные результаты в виде извлеченной информации достаточно представить в файле с форматом .csv для удобной загрузки в базу данных.

4. Подготовка источников

Для работы с большим объемом данных в качестве корпуса для извлечения семантических отношений необходимо провести анализ исследуемой литературы.

4.1. Задачи унификации данных источников

После оцифровки источников каждого документа необходима отдельная обработка для унификации данных:

1. Конвертация источника в формат с расширением .txt.
2. Объединение нескольких текстовых файлов источника в один файл.
3. Удаление пустых строк в файле.
4. Удаление диакритик, знаков и символов с неправильной кодировкой.
5. Удаление номеров страниц, слогов, показывающих начало терминов, опечаток и др.
6. Скрипты написаны на языке Python, который подходит для обработки русского языка за счет большого выбора библиотек. В данном случае, была использована библиотека «re (Regular expression operations)».

4.2. Соотношение терминов и определений в источниках

Источник КЛЭ+ЛЭ представляет собой файл с расширением .txt размером 85 МБ и количеством строк 295 928. После очистки было удалено 64 889 строк, которые содержали символы, обозначающие номера страниц, ссылки на иллюстрации и библиографию. Библиография была удалена из определений, поскольку данная информация не пригодится для извлечения отношений между терминами и может помешать процедуре автоматического заполнения полей.

После анализа СЛТ было установлено, что из 286 терминов, совпадающих со словником, 38 терминов СЛТ не имеют определения, а имеют отсылку на другой термин, например, «Ускорение — см. Пиррихий». Определения терминов в данном источнике записаны в виде развернутого ответа, средняя длина определения составляет 5 996 символов, что занимает порядка 70 строк.

В таблице 1 представлены характеристики исследуемых источников, а именно: название, год издания, количество уникальных терминов в источнике, количество полноценных определений без учета ссылок на другие и краткое описание самого источника.

5. Анализ терминов

В словнике, составленном экспертами предметной области, представлено 1 544 уникальных термина, составленных вручную.

В таблице 2 приведена статистика встречаемости терминов из словника в источниках, объявленных ранее. Таким образом, в каждом из источников встретилось 128 терминов, представленных в словнике, данные термины являются общеупотребительными. 587 терминов не встретилось ни в одном источнике. Из КЛЭ был извлечен 571 термин, что составляет порядка 3,7% определений, использованных из всего источника. 401 термин встретился в ЛЭ, что составляет порядка 8,3% от всего

Таблица 1. Описание и характеристики исследуемых источников

Источник	Год издания	Кол-во терминов	Кол-во определений	Краткое описание
КЛЭ	1962–1978	15 228	14 755	Наличие большого количества терминов в источнике подразумевает отдаленность от предметной области. В данной энциклопедии собран материал по персоналиям и более общим терминам, частично связанным с поэтологией
ЛЭ	1929–1939	4 782	4 276	Источник содержит полные определения, порядка 30–60 строк в среднем
СЛТ	1925	739	679	Словарь представляет общеобразовательное пособие из области теории литературы, лингвистической поэтики, эстетики и социологии художественного творчества
ПСК	1966	673	615	Поэтический словарь содержит максимально приближенные термины к предметной области. Определения характеризуются небольшим размером и очевидным сходством в построении определений
БСЭ	1969–1978	30 томов	–	Источник использовался в качестве вспомогательного

Таблица 2. Статистика частоты появления термина в словнике и в источниках

Название источника	Количество терминов из словника	Количество терминов в источнике	Соотношение
КЛЭ	571	15 228	3,7%
ЛЭ	401	4 782	8,3%
СЛТ	286	739	38%
ПСК	601	673	89%
Ни в одном источнике	587	–	–
Во всех источниках	128	–	–

Таблица 3. Статистика слов в терминах из словника

Количество слов в термине	Количество терминов	Соотношение со словником
1	996	64,5%
2	478	31%
3	60	3,9%
4	6	0,39%
5	3	0,2%
6	1	0,01%

объема ЛЭ. 286 терминов было извлечено из СЛТ, что составляет порядка 38% от всего объема СЛТ. Наибольшbv по объему извлеченных терминов и определений оказался источник ПСК – извлечен 601 термин из 673, что составляет 89%.

В таблице 3 исследованы термины из словника на количество слов в термине. Данная количественная мера важна для дальнейшего исследования, поскольку автоматические извлечение семантических отношений обычно проверяется на терминах длины не более чем 1. В данном случае, количество однословных терминов составляет 64,5% от всего количества терминов в словнике, второе место занимают двухсловные термины – 31%, третье место – 3,9% – трехсловные термины. Кроме того, в словнике присутствует шестисловный термин в единичном экземпляре и 3 пятисловных термина.

6. Методика поиска термина и определения

Следующим шагом служит выделение термина и его определения из источника. Задача является нетривиальной, поскольку компьютер различает поступающий текст только в виде строки, необходимо наложить ряд правил и условий для корректного извлечения терминов и определений.

Для этого необходимо разбить задачу нахождения термина и определения на подзадачи:

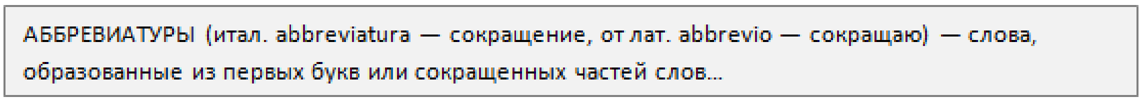
1. Поиск начала термина;

2. Поиск конца термина;

3. Поиск начала определения;

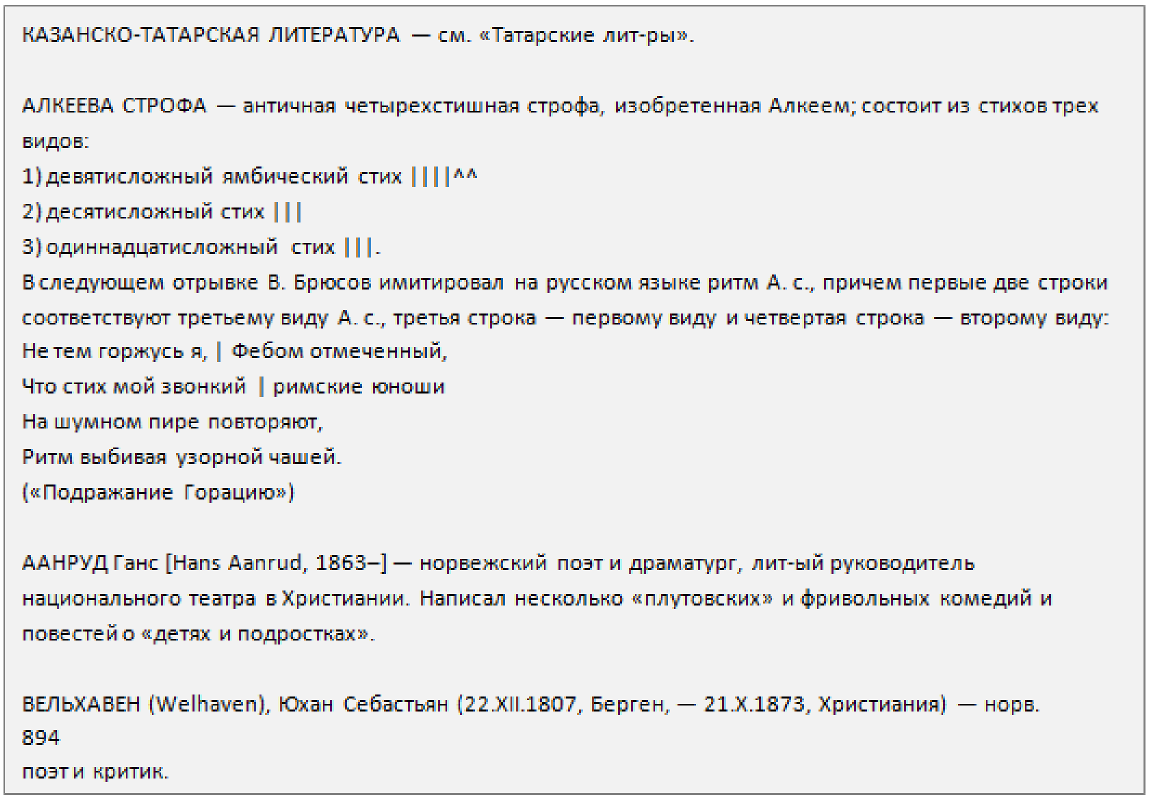
4. Поиск конца определения.

Разбиение задачи поиска термина на две подзадачи (поиск начала и поиск конца термина) связано с тем, длина термина не фиксирована: термин может быть однословным, но может состоять и из нескольких слов. Начало определения не всегда совпадает с концом термина или опознавательным знаком, используемых в словарях, а именно тире. Между термином и определением можно встретить конструкции в круглых и/или квадратных скобках, в которых добавлена дополнительная информация:


АББРЕВИАТУРЫ (итал. abbreviatura — сокращение, от лат. abbrevio — сокращаю) — слова, образованные из первых букв или сокращенных частей слов...

Определение может состоять из одной или нескольких строк. Точка или символ конца строки не являются показателями окончания определения. Маркером окончания определения может служить начало следующего термина. Важно понять, где начинается следующий термин, так как термин может встретиться и в самом определении в роли поясняющего понятия. В определении часто можно встретить в качестве примеров стихотворения, цитаты, отсылки на другие термины. Определения могут состоять как из одного предложения, так и из нескольких десятков строк. Ниже продемонстрировано два термина с определениями. Первый термин состоит из трех слов и в качестве определения имеет отсылку на другой термин. Второй термин состоит из двух слов и имеет в определении виды Алкеевой строфы и пример – отрывок из произведения «Подражание Горацию».

Прежде чем разработать правила для алгоритма автоматического извлечения терминов и определений, необходимо изучить исследуемые источники. Каждый словарь имеет свою уникальную специфику интерпретации термина. Важно разработать совокупность правил, которые позволили бы с большой точностью и полнотой определять начало и конец терминов и определений. Для данной задачи нет смысла подключать методы машинного обучения, так как при использовании шаблонов с регулярными выражениями и ряда правил получился качественный результат, который стал достижим за несколько итераций.



КАЗАНСКО-ТАТАРСКАЯ ЛИТЕРАТУРА — см. «Татарские лит-ры».

АЛКЕЕВА СТРОФА — античная четырехстишная строфа, изобретенная Алкеем; состоит из стихов трех видов:
1) девятисложный ямбический стих ||||^^
2) десятисложный стих |||
3) одиннадцатисложный стих |||.
В следующем отрывке В. Брюсов имитировал на русском языке ритм А. с., причем первые две строки соответствуют третьему виду А. с., третья строка — первому виду и четвертая строка — второму виду:
Не тем горжусь я, | Фебом отмеченный,
Что стих мой звонкий | римские юноши
На шумном пире повторяют,
Ритм выбивая узорной чашей.
(«Подражание Горацию»)

ААНРУД Ганс [Hans Aanrud, 1863–] — норвежский поэт и драматург, лит-ый руководитель национального театра в Христиании. Написал несколько «плутовских» и фривольных комедий и повестей о «детях и подростках».

ВЕЛЬХАВЕН (Welhaven), Юхан Себастьян (22.XII.1807, Берген, — 21.X.1873, Христиания) — норв.
894
поэт и критик.

Таким образом, можно выделить несколько правил, выведенных эмпирическим путем, которые помогут извлечь из источника термин и его определение:

1. Новая строка может начинаться с термина.
2. Термин может быть написан заглавными буквами.
3. Термин может быть разделен символами и знаками, не относящимися к самому термину.
4. Слова (или слово), входящее в термин, имеет начальную форму (стоит в им.п., ед.ч.)
5. Термин может быть отделен от прилагательного знаком тире, равно или пробелом.
6. Между термином и определением может стоять в круглых скобках перевод термина с другого языка.
7. Между термином и определением может стоять в квадратных скобках поясняющая информация.
8. Термин может обозначать личность, т.е. в качестве термина будет фамилия, отделенная запятой от имени или другой информации, которая может быть расположена в скобках.

Таблица 4. Статистика слов в терминах из словника

Количество определений	Количество терминов
1	446
2	249
3	134
4	118
5	1

9. В используемых источниках термин, смысл которого раскрывается, в определении встречается в виде сокращения первой буквы с точкой. Если термин состоит из двух или более слов, тогда сокращение имеет вид перечисления первых букв разделенных точкой и пробелом (например, Алкеева строфа = А. с.).

В таблице 4 приведены количественные характеристики после извлечения всех определений из исследуемых источников, таким образом, 446 терминов имеют только 1 определение, 249 терминов имеют по 2 определения, 134 термина имеют по 3 определения и так далее.

7. Распознавание семантических отношений

Люди имеют уникальную способность определять взаимосвязь между двумя лексическими единицами, однако оценить близость двух терминов с помощью программного комплекса возможно с помощью метрик семантической близости. Метрики семантической близости являются неким индикатором для определения взаимосвязи между терминами на основе задаваемого правила. В работе [19] рассмотрены метрики близости для установки отношений между словами из статей Википедии.

7.1. Метрика «Количество общих слов в определении»

Метрика (1) использует меру семантической близости на основе общих слов определений двух терминов.

$$similarity(t_i, t_j) = \frac{2|(d_i \cap d_j)/stopwords|}{|d_i| + |d_j|} \tag{1}$$

Числитель дроби равен количеству общих слов, приведенных в начальную форму, в выбранных определениях с учетом списка слов, определяемых как стоп-слова. Знаменатель дроби соответствует сумме всех слов в каждом из двух выбранных определений.

В качестве источника стоп-слов был выбран «Частотный словарь современного русского языка» [23]. Удаление стоп-слов помогает снижать уровень шума, другими словами, повышается качество выбранной метрики, поскольку, например, союз «и» не несет никакой смысловой нагрузки в определении для нахождения взаимосвязи двух терминов, однако может встретиться в обоих определениях.

Однако данная метрика не учитывает длину определений. Это является главным недостатком метрики, поскольку длина некоторых определений может достигать до ста строк и иметь большое количество общеупотребительных терминов, которые не относятся к списку стоп-слов, например: система, совокупность, ряд, количество и т.д.

7.2. Метрика «Косинусная мера сходства», или «Косинус угла между векторами определений»

Для того чтобы компенсировать влияние длины определений на связность между терминами, применяется метрика под названием косинусная мера сходства. Определение представляется как N-мерный вектор, и далее происходит оценка терминов с использованием полученных векторов.

Размерность вектора определяется в зависимости от характера исследуемой задачи. Для данного случая размерность вектора определения совпадает с количеством слов в словнике, а именно 1554. Таким образом, вектор определения будет включать только термины из словника. Если термин из словника не встретился в определении, то обозначенный элемент становится равным нулю, в других случаях элемент становится равным количеству появлений термина в определении.

В формуле косинусной меры сходства (2) числитель представляет собой скалярное произведение векторов двух рассматриваемых определений, а знаменатель равен произведению евклидовых норм этих векторов. Знаменатель в формуле нормирует по длине векторы, таким образом, результат можно интерпретировать как скалярное произведение нормированных векторов, соответствующих двум определениям.

Для реализации метрики необходимо провести предварительные процедуры:

1. Нормализация определений — приведение каждого слова определения в начальную форму;
2. Нормализация словника — приведение каждого слова термина из словника в начальную форму;
3. Поиск терминов в определении (поиск производится с учетом всех слов в терминах из словника);
4. Формирование векторов определений размерности, равной количеству терминов в словнике.

Минимальная близость терминов соответствует количественному показателю 0.0, а максимальная – 0.99.

$$similarity(t_i, t_j) = \frac{f_i \cdot f_j}{||f_i|| \cdot ||f_j||} = \frac{\sum_{k=1,N} f_{ik} f_{jk}}{\sqrt{\sum_{k=1,N} f_{ik}^2}\sqrt{\sum_{k=1,N} f_{jk}^2}} \tag{2}$$

где f_{ik} — частота леммы c_k в определении d_i [8].

8. Описание алгоритма

Входными данными служит множество терминов T, между элементами которого необходимо установить семантические отношения по их определениям из множества D. Таким образом, задача сводится к задаче распознавания множества семантических отношений R из всех возможных пар терминов. Например:

$$T = \{\text{стопа,новелла,единоначатие,ямб,анафора}\}$$

$$R = \{<\text{стопа,ямб}>,<\text{единоначатие,анафора}>\}$$

9. Сравнение метрик

Для интерпретации полученных результатов, необходимо использовать ручную оценку. Были выбраны случайным образом 26 пар терминов. В качестве асессора выступает эксперт предметной области. Ниже приведены нормированные результаты метрики «количество общих слов в определении» — «Метрика 1» и метрики «косинусная мера угла» — «Метрика 2», диапазон результатов по двум метрикам и оценке эксперта составляет от 0 до 0.99. В таблице 5 представлено сравнение двух метрик близости терминов с ручной оценкой эксперта.

10. Заключение

В работе представлены методы по автоматическому распознаванию взаимосвязи терминов путем создания алгоритма на основе семантических метрик близости. Метрика «Количество общих слов в определении» является универсальным и наиболее очевидным способом установления взаимосвязи терминов по определениям, однако данный метод зависит от длины определений, что является важным фактором ввиду специфики предметной области. Напротив, метрика «Косинус угла между векторами определений» не учитывает длину определений, так как в основу заложен набор слов, задаваемых в качестве векторного представления. В данном случае длина вектора совпадала с количеством терминов словника, что позволило детектировать общие термины из словника в рассматриваемых определениях. Для формирования данных была проведена обработка словарей предметной области.

Таблица 5. Сравнение двух метрик близости терминов с ручной оценкой эксперта

№	Термины	Метрика 1	Метрика 2	Оценка эксперта
1	АНТИБАКХИЙ, ЖЕЛЬДИРМЕ 1. Античный размер. 2. Казахский размер.	0.00	0.00	0.00
2	ПЯТИДОЛЬНИК, ЭПИГРАФ 1. Пятисложник, силлаб.-тоническ. размер. 2. Жанр литературы.	0.01	0.00	0.00
3	ХОРЕЙ, КАНТИЛЕНА 1. Стихотворный размер. 2. Музыкально-поэтический жанр.	0.01	0.22	0.05
4	ДОХМИЙ, КРАТА 1. Античный размер. 2.Обобщение равносложн. стоп в силлабо-тонических размерах.	0.01	0.32	0.00
5	ГЕКЗАМЕТР, КОБЗАРЬ 1.Античный размер, в русской метрике имитируется 6-стопным дактилем. 2. Певец народных украинских песен.	0.02	0.00	0.00
6	ШЕСТИДОЛЬНИК, ТОНИЧЕСКОЕ СТИХОСЛОЖЕНИЕ 1. 6-сложная 2-акцентная стопа в силлабо-тонике. 2. Изначальное название силлабо-тоники.	0.03	0.22	0.20
7	ДОЛГИЙ СЛОГ, АРУЗ 1. В античной и некоторых современных метриках. 2. Метрика на основе чередования долгих и кратких слогов.	0.04	0.72	0.50
8	БЫЛИНЫ, МАКАРОНИЧЕСКИЕ СТИХИ 1. Жанр русского народного эпоса. 2. Жанр сатирических стихов.	0.05	0.81	0.10
9	ЧАСТУШКА, ПОСЛОВИЦА 1. Жанр лирической поэтической формы фольклора. 2. Жанр краткой поэтической формы фольклора.	0.06	0.60	0.50
10	ПАУЗНИК, ЭВРИТМИЯ 1. Стихи с неполносложной 3-сложной стопой. 2. Благозвучие стиха.	0.07	0.77	0.05

№	Термины	Метрика 1	Метрика 2	Оценка эксперта
11	АНАФОРА, ЦЕНТОН 1. Звуковой, лексический, синтаксический повтор в начале смежных стихов или строф. 2. Составление стихотворений из различных известных стихов.	0.08	0.99	0.00
12	СТОПА, ПЭОН 1. В силлабо-тонике повторяемая группа слогов с акцентом на заданном месте (род). 2. В силлабо-тонике 4-сложная стопа с акцентом на заданном месте (вид).	0.13	0.76	0.95
14	ПИРРИХИЙ, МОЛОСС 1. Пропуск ударения в 2-сложной стопе, нарушающий правильность ритмической схемы. 2. 3 ударения (2 сверхсхемных) в 3-сложной стопе (тримакр).	0.15	0.36	0.10
15	ИПОСТАСА, ДИАСТОЛА 1. Ритмическая модификация метрической стопы. 2. В античной метрике замена долгого слога кратким.	0.29	0.62	0.00
16	ТРОХЕЙ, ДОХМИЙ 1. Античная стопа. 2. Античный размер.	0.33	0.49	0.10
17	ЭПИТРИТ, ИОНИК 1. Античная 4-сложная стопа (3 долгих и 1 краткий). 2. Античная 4-сложная стопа (2 долгих и 2 кратких).	0.42	0.82	0.50
18	ДИХОРЕЙ, ТРИМЕТР 1. Диподия, четырехсложная стопа с ударением на 3-м или 4-м слоге (пэон 3-й или 4-й) (часть). 2.В античной метрике размер из 3 диподий (целое).	0.46	0.90	0.50
19	АНТИБАКХИЙ, БРАХИХОРЕЙ 1. Античная 3-сложная стопа (2 долгих и 1 краткий). 2. 3-сложная стопа (краткий, долгий и краткий), амфибрахий.	0.53	0.18	0.50

№	Термины	Метрика 1	Метрика 2	Оценка эксперта
20	ПЕНТАМЕТР, ДИСТИХ 1.В русской метрике античный 5-стопный дактилический стих имитируется 6-стопным с мужской цезурой (часть). 2. Элегический дистих, двустишие из гекзаметра и пентаметра (целое).	0.68	0.96	0.80
21	ДИЯМБ, ДИХОРЕЙ 1. Двойная ямбическая стопа (диподия). 2. Двойная хореическая стопа (диподия).	0.99	0.86	0.70

Список литературы / References

[1] Лингвистическая онтология «Тезаурус Рутез», http://www.labinform.ru/pub/ruthes/.

[2] Тезаурус WordNet, http://wordnet.princeton.edu/.

[3] Бойков В. Н. и др., “Тезаурус как инструмент поэтологии”, *Моделирование и анализ информационных систем*, **17**:1 (2010), 5–24; [Boikov V. N. et al., “Thesaurus as a poetological tool”, *Modeling and Analysis of Information Systems*, **17**:1 (2010), 5–24, (in Russian).]

[4] Бойков В. Н., “Семантическая модель «Тезауруса по поэтологии» в составе информационно-аналитической системы”, *Материалы научной конференции «Интернет и современное общество»*, 2013, 273–279; [Boikov V. N., “Semanticheskaya model «Tezaurusa po poehtologii» v sostave informacionno-analiticheskoj sistemy”, *Materialy nauchnoj konferencii «Internet i sovremennoe obshchestvo»*, 2013, 273–279, (in Russian).]

[5] Бойков В. Н., “Предметно-ориентированный тезаурус в открытой информационно-аналитической системе”, *RCDL’2013 «Электронные библиотеки: перспективы, методы и технологии, электронные коллекции»*, 2013, 70–76; [Boikov V. N., “Predmetno-orientirovannyj tezaurus v otkrytoj informacionno-analiticheskoj sisteme”, *RCDL’2013 «Ehlektronnye biblioteki: perspektivy, metody i tekhnologii, ehlektronnye kollekcii»*, 2013, 70–76, (in Russian).]

[6] Тезаурус по поэтологии, http://wikipoetics.ru/.

[7] Hearst M. A., “Automated discovery of WordNet relations”, *WordNet: an electronic lexical database*, 1998, 131–153.

[8] Панченко А. И., “Извлечение семантических отношений из статей Википедии с помощью алгоритмов ближайших соседей”, *Открытые системы*, **16** (2012), 18–27; [Panchenko A. I., “Izvlechenie semanticheskih otnoshenij iz statej Vikipedii s pomoshchyu algoritmov blizhajshih sosedej”, *Otkrytye sistemy*, **16** (2012), 18–27, (in Russian).]

[9] Serelex: Поиск семантически связных слов, http://serelex.org/ru.

[10] Киселев Ю. А., “Метод извлечения родовидовых отношений между существительными из определений толковых словарей”, *Программная инженерия*, **10** (2015), 38–48; [Kiselev YU. A., “Metod izvlecheniya rodovidovyh otnoshenij mezhdu sushchestvitelnymi iz opredelenij tolkovyh slovarej”, *Programmnaya inzheneriya*, **10** (2015), 38–48, (in Russian).]

[11] *Краткая литературная энциклопедия*, Сов. Энцикл., 1962–1978; [*Kratkaya literaturnaya ehnciklopediya*, Sov. Ehncikl., 1962–1978, (in Russian).]

[12] *Литературная энциклопедия*, Ком. акад., 1929–1939; [*Literaturnaya ehnciklopediya*, Kom. akad., 1929–1939, (in Russian).]

[13] *Словарь литературных терминов*, Изд-во Л. Д. Френкель, 1925; [*Slovar literaturnyh terminov*, Izd-vo L. D. Frenkel, 1925, (in Russian).]

[14] Квятковский А. П., *Поэтический словарь*, Сов. Энцикл, 1966; [Kvyatkovskij A. P., *Poehticheskij slovar*, Sov. Ehncikl, 1966, (in Russian).]

[15] *Большая советская энциклопедия*, Сов. энцикл., 1969–1978; [*Bolshaya sovetskaya ehnciklopediya*, Sov. ehncikl, 1969–1978, (in Russian).]

[16] Зализняк А. А., *Грамматический словарь русского языка*, Словоизменение, 1980; [Zaliznyak A. A., *Grammaticheskij slovar russkogo yazyka*, Slovoizmenenie, 1980, (in Russian).]

[17] Mystem, https://tech.yandex.ru/mystem/.

[18] Segalovich I., "A Fast Morphological Algorithm with Unknown Word Guessing Induced by a Dictionary for a Web Search Engine", *MLMTA*, 2003, 273–280.

[19] PyMystem, https://pypi.python.org/pypi/pymystem3/0.1.1.

[20] PyMorphy2, https://pymorphy2.readthedocs.org/en/latest/.

[21] OpenCorpora, http://opencorpora.org/.

[22] FrameWork Django, https://www.djangoproject.com/.

[23] Ляшевская О. Н., *Частотный словарь современного русского языка (на материалах Национального корпуса русского языка)*, Азбуковник, 2009; [Lyashevskaya O. N., *Chastotnyj slovar sovremennogo russkogo yazyka (na materialah Nacionalnogo korpusa russkogo yazyka)*, Azbukovnik, 2009, (in Russian).]

DOI: 10.18255/1818-1015-2015-6-834-851

Linguistic and Statistical Analysis of the Terminology for Constructing the Thesaurus of a Specified Field

Karyaeva M. S.[1]

Received May 15, 2015

The paper is devoted to the analysis of the body of terms and terminological sources for further automation of constructing the thesaurus of a subject area, which is regarded as poetics in our work. Preliminary systematization of terminology with a linguistic and statistical approach forms the body of semantically related concepts to automate extraction of semantic relationships between terms that define the structure of the thesaurus of the specified field.

Keywords: thesaurus, semantic similarity metrics, data mining, computer linguistics

For citation: Karyaeva M. S, "Linguistic and Statistical Analysis of the Terminology for Constructing the Thesaurus of a Specified Field", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 834–851.

On the authors:
Karyaeva Maria Sergeevna, orcid.org/0000-0003-4466-1735, graduate student,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: mari.karyaeva@gmail.com

Acknowledgments:
[1]This work was supported by Russian Foundation for Basic Research RFBR Project 13-06-00448

Моделирование и анализ информационных систем. Т. 22, №6 (2015), с. 852–861
Modeling and Analysis of Information Systems. Vol. 22, No 6 (2015), pp. 852–861

DOI: 10.18255/1818-1015-2015-6-852-861

УДК 004.415.25

Разработка активного внешнего модуля сетевой топологии для контроллера программно-конфигурируемой сети Floodlight

Носков А. А., Никитинский М. А., Алексеев И. В.

получена 15 декабря 2015

Традиционная архитектура сети передачи данных является негибкой и сложной. Данное обстоятельство привело к появлению парадигмы программно-конфигурируемой сети (ПКС), в которой уровень управления сетью отделен от уровня передачи данных. Это стало возможно за счет переноса плоскости управления с коммутационного оборудования в программные модули, которые работают на выделенном сервере, называемом контроллером (или сетевой операционной системой), или в сетевые приложения, которые работают с этим контроллером. Способы представления, хранения и интерфейсы взаимодействия с элементами сетевой топологии, доступные пользователям контроллера ПКС, являются одними из наиболее важных аспектов сетевых операционных систем. Данное обстоятельство обусловлено тем, что функционирование некоторых ключевых модулей контроллера в существенной степени основано на внутреннем представлении сетевой топологии. Такими модулями, к примеру, являются модуль firewall, модуль маршрутизации и т.д. В данной статье рассмотрены применяемые способы представления и хранения сетевой топологии, а также интерфейсы взаимодействия с соответствующими модулями контроллера Floodlight. Предложен и разработан альтернативный алгоритм обмена сообщениями об изменении сетевой топологии между контроллером и сетевыми приложениями, позволяющий реализовать оповещение на основе подписки на соответствующие события. Разработан API для модуля взаимодействия с прикладными программами контроллера программно-конфигурируемой сети. На основе данного алгоритма и API разработан модуль Topology Tracker, способный в активном режиме сообщать сетевым приложениям о произошедших изменениях в топологии сети и хранящий ее компактное представление для ускорения процесса взаимодействия.

Ключевые слова: программно-конфигурируемая сеть, контроллер Floodlight, внешний модуль, сервис, ПКС, сетевая топология, Topology Tracker, DEventBus, Link Discovery

Для цитирования: Носков А. А., Никитинский М. А., Алексеев И. В., "Разработка активного внешнего модуля сетевой топологии для контроллера программно-конфигурируемой сети Floodlight", *Моделирование и анализ информационных систем*, **22**:6 (2015), 852–861.

Об авторах:
Носков Андрей Александрович, orcid.org/0000-0002-2268-4912, инженер,
ООО «Энергия-Инфо», ул. Союзная, 144, г. Ярославль, 150008 Россия, e-mail: naa@a-real.ru

Никитинский Михаил Александрович, orcid.org/0000-0001-8830-8613, программист-аналитик,
ООО «Энергия-Инфо», ул. Союзная, 144, г. Ярославль, 150008 Россия, e-mail: man@a-real.ru

Алексеев Игорь Вадимович, orcid.org/0000-0001-8321-2399, канд. физ.-мат. наук, директор центра Интернет,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия, e-mail: aiv@yars.free.net

Благодарности:
Работа выполнена в ФГБОУ ВПО "Ярославский государственный университет им. П.Г. Демидова" при финансовой поддержке Министерства образования и науки Российской Федерации в рамках Соглашения о предоставлении субсидии (*ID RFMEFI57414X0036*).

Введение

Одной из наиболее заметных тенденций в развитии сетевых технологий в наши дни является подход разделения уровней управления сетью и передачи данных за счет переноса управляющих воздействий с коммутационного оборудования (gateway, switch, router, hub и т.д.) в программные модули, которые работают на выделенном сервере, называемом контроллером, или в сетевые приложения, которые работают с контроллером. Разделение уровней управления и передачи в коммутационном оборудовании связано с тем, что оборудование, выпускаемое различными компаниями, является закрытым с точки зрения реализации и организации. Практически каждое устройство — это определенная система, в которой есть физическая составляющая, есть конфигурационная система и есть набор приложений. Данная организация сетевых устройств усложняет их взаимодействие, а введение нового устройства или приложения в сетевую инфраструктуру порой является весьма нетривиальной задачей [1]. Для решения данной проблемы был предложен новый подход к организации взаимодействия сетевого оборудования — подход ПКС [2]. Основная идея такого подхода была сформулирована специалистами университетов Стэнфорда и Беркли в 2006 году. Произведенные ими исследования нашли поддержку не только в университетах по всему миру, но и были позитивно восприняты более чем четырьмя десятками ведущих производителей сетевого оборудования, такими как CISCO SISTEMS, International Business Machines, Hewlett-Packard, Nippon Electronics Corporation.

Коммуникационная сеть называется ПКС, если содержит как минимум один контроллер сети, с установленной сетевой операционной системой, и как минимум один аппаратный или виртуальный OpenFlow-коммутатор, при этом в состав сети могут входить другие коммутационные устройства. В большинстве случаев при организации ПКС разработчики связывают контроллер с коммутационным оборудованием с помощью протокола OpenFlow [3], перенося логическую составляющую с коммутационных устройств на контроллер. Контроллер управляет OpenFlow-коммутаторами по выделенному защищенному каналу связи. На него возлагается ответственность за решения по маршрутизации сетевого трафика, управление безопасностью, разведку топологии сети и прочее. Данный функционал реализуется при помощи внутренних модулей контроллера или сетевых приложений, устанавливаемых на него.

Что касается сетевых приложений, то разработчики отмечают здесь проблему отсутствия единого стандартизированного *API* (application programming interface [4]) между сетевыми приложениями и контроллером, а также невозможность использовать приложение одного контроллера на другом [5]. Это связано с тем, что контроллеры написаны на различных языках программирования и имеют различный базовый функционал. При этом приложения, запущенные на одном контроллере, работают консистентно — они располагают одинаковой информацией о состоянии сети. Таким образом, задача управления сетью теперь возложена на разработчиков контроллеров и приложений для них, что фактически создает новый рынок — рынок сетевых приложений для контроллеров. Например, ряду приложений требуется решать задачу прокладки оптимальных маршрутов между конечными узлами в сети. Так как сеть может быть представлена в виде графа, в котором ребра — соединение между сетевыми узлами, а вершины соответствуют этим узлам, то для поиска оптимального пути между конечными узлами может быть применена теория графов. Для построения оптимального маршрута может быть использован алгоритм Дейкстры [6], но если веса ребер изменяются часто или необходимо учитывать дополнительные параметры сети (такие как политики *Quality of Service* [7] или требования безопасности), то использование алгоритма Дейкстры для полного перерасчета кратчайших путей может быть нецелесообразным [8]. Пример различий в реализациях сетевых операционных систем можно найти и в способах обмена информацией с сетевыми приложениями. Классический подход в организации взаимосвязи такого рода предполагает использование протокола TCP [9], в качестве альтернативного подхода было предложено использование асимметричного протокола [10, 11].

Одними из наиболее востребованных модулей контроллера, в любом существующем на сегодняшний день контроллере ПКС, являются модули построения сетевой топологии. Данные модули собирают информацию о коммутаторах, хостах и связи между ними, а также отвечают за периодическое обновление информации о топологии. Обычно разработчики контроллеров называют эти модули *discovery* и *topology*, реже они объединены в один модуль. Сетевые приложения, работающие с контроллером ПКС, могут запрашивать информацию о текущем состоянии сети, но, как правило, не имеют возможности подписываться на получение сообщений о динамическом изменении топологии. Данное обстоятельство приводит либо к некорректной работе приложений, либо к необходимости периодически обращаться сетевым приложениям к контроллеру.

1. Контроллер Floodlight

Одним из наиболее динамично развивающихся контроллеров, поддерживаемых сообществом разработчиков во всем мире, является контроллер *Floodlight* [12]. FloodLight — контроллер корпоративного уровня, написан на языке Java, имеет лицензию Apache, разрабатывается компанией Big Switch Networks и является ядром для платного контроллера Big Cloud Fabric SDN controller. На момент написания статьи текущая версия контроллера FloodLigth 1.1. Контроллер имеет модульную структуру, за счет которой облегчается процесс расширения и внесения изменений, поддерживает широкий спектр виртуальных и физических коммутаторов, способен поддерживать смешанные OpenFlow сети и сети традиционной архитектуры. При описании архитектуры контроллера используют четыре основных понятия: сервисы, внутренние и внешние модули, сетевые приложения. Сервис — это интерфейс, который экспортирует состояние и генерирует события. Потребители сервиса могут получать/устанавливать состояние и подписываться или отписываться от событий. При этом допускается множество реализаций одного и того же сервиса. Каждый модуль может использовать некоторый набор сервисов для реализации некоторой функциональности. Модуль может предоставлять соответственно ноль или более сервисов. Все модули в FloodLight имеют минимальное количество зависимостей между собой, что упрощает разработку приложений. Различие между внешними и внутренними модулями заключается в том, что без внешних модулей контроллер сможет работать, а без внутренних нет. Сетевое приложение — это программа, которая по *REST API* [13] взаимодействует с внутренними и внешними модулями, может устанавливаться как на самом контроллере, так и удаленно. Общая структура контроллера Floodlight представлена на рис. 1.

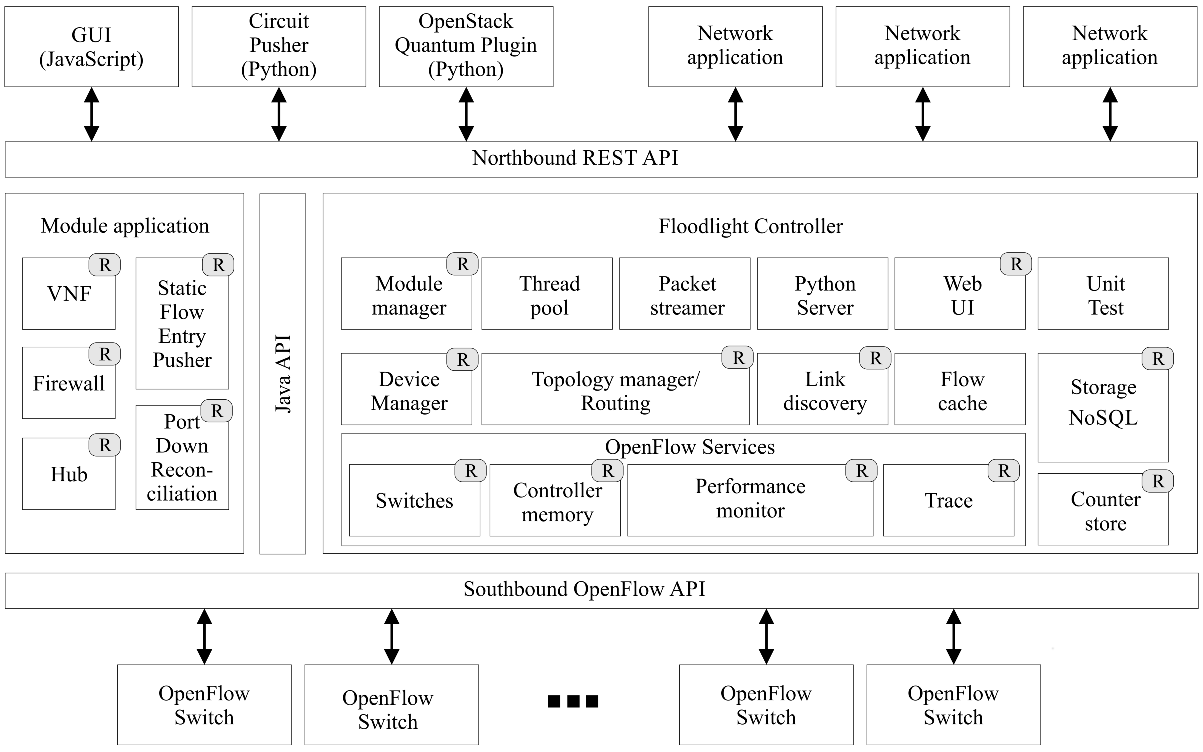


Рис. 1. Общая структура контроллера Floodlight

Контроллер Floodlight использует для разведки топологии модуль *Link Discovery*. Данный модуль работает с протоколом *LLDP (Link Layer Discovery Protocol)* [14]. LLDP является протоколом канального уровня, который позволяет сетевому оборудованию уведомлять о своем существовании другие узлы сети, а также обрабатывать подобные сообщения, приходящие от других устройств.

Информация, полученная контроллером Floodlight из LLDP Data Unit сообщений, сохраняется на устройстве в виде *management information base (MIB)* [15–17]. Таким образом, общая топология сети получается путем сбора и объединения информации со всех устройств. Собираемая информация может содержать следующие данные:

- имя устройства и его описание,
- имя порта и его описание,
- имя VLAN,
- IP-адрес управления устройством через протокол SNMP,
- функции, которые может выполнять устройство,
- информация о MAC/PHY,

- информация о питании через Ethernet (Power over Ethernet),
- информация об объединении каналов.

Используя LLDP, контроллер Floodlight, при включении получает информацию обо всех OF-коммутаторах [18], присутствующих в сети, и при этом не имеет информации о конечных узлах сети. Эта информация появляется по мере работы конечных узлов, как только они начинают пересылать какой-либо трафик через сеть, информация о них записывается в MIB.

Однако модуль Link Discovery не может отправлять динамические изменения, происходящие в сети, сетевым приложениям. Данная особенность оказывает серьезное влияние на работу сетевых приложений. Например, за графическое отображение в web-браузере топологии ПКС в контроллере Floodlight отвечает сетевое приложение GUI. Для отображения актуального состояния ПКС в приложении GUI необходимо либо дать команду актуализировать информацию на данный момент, либо установить требование постоянно обновлять информацию о состоянии сети, при этом время обращения к контроллеру фиксировано, неизменно и равно трем секундам.

2. Принципы функционирования внешнего модуля Topology Tracker

При изменении сетевой структуры ПКС в модуль Link Discovery поступают сообщения, содержащие информацию об источнике и параметрах произошедшего изменения. Внутренние модули контроллера обрабатывают их, соответствующим образом изменяя внутреннее представление сети. Затем через Java API подписанным внешним модулям контроллера передается информация о произошедших изменениях. Внешние модули могут формировать некоторую реакцию на эти события.

В статье [19] авторами был предложен механизм взаимосвязи сетевых приложений с контроллером ПКС. Данный механизм обеспечивает сетевые приложения возможностью осуществлять подписку на предоставляемые сервисы как контроллером, так и другими сетевыми приложениями.

Так как внешние модули могут быть подписаны на изменения, происходящие в сетевой инфраструктуре, а изменение ядра контроллера Floodlight может привести к критическим ошибкам, был разработан внешний модуль Topology Tracker. Основными функциями данного модуля является собственное представление сетевой топологии и передача информации сетевым приложениям, которые подписаны на произошедшие изменения в ПКС. Topology Tracker формирует представление сетевой топологии в виде трех ассоциативных массивов для хранения информации о маршрутизаторах, конечных устройствах и соединениях соответственно. В первых двух случаях ключом является идентификатор устройства (в простейшем случае – MAC-адрес устройства). Ключом для массива соединений является строка, образованная конкатенацией строковых представлений идентификаторов конечных устройств. Значениями, во всех случаях, являются специальные классы представлений, организованные таким образом, чтобы уменьшить время поиска запрашиваемого устройства и его параметров. Минусом такой структуры является относительно медленное добавление и удаление элементов массивов.

Модуль Topology Tracker может взаимодействовать с внутренними модулями контроллера Floodlight через Java API для восстановления информации о состоянии сети при обнаружении некорректных данных, а также в случае остановки и возобновления работы. Когда происходит изменение в топологии сети, контроллер по очереди предоставляет данную информацию внутренним модулям. Модуль Topology Tracker получает данные после обработки их модулем Link Discovery. Обработав поступившую информацию, модуль Topology Tracker отправляет сформированные данные на внутренний сокет контроллера Floodlight. Данный сокет прослушивает локальное сетевое приложение DEventBus, которое выступает в роли сетевой шины с механизмом регистрации событий от модулей и приложений, а также с возможностью подписки на данные события. Приложение DEventBus было разработано в соответствии с логикой, описанной в [19]. Если приложению DEventBus необходимо передать данные на удаленное устройство, на котором находится подписанное на событие сетевое приложение, приложение DEventBus связывается с аналогичным приложением на стороне приема и через него передает данные. Если сетевое приложение находится локально, то от модуля DEventBus ему передаются данные. Схема обмена сообщениями об изменениях сетевой топологии ПКС представлена на рис. 2. Сплошной линией обозначен предлагаемый метод обмена сообщениями, пунктирной линией – метод обмена сообщениями, применяемый в контроллере Floodlight в настоящее время. Штрих-пунктирной линией обозначена защищенная схема обмена сообщениями при запросе, адресованном через REST API разработанному модулю. Во всех случаях стрелками обозначено направление передачи сообщений.

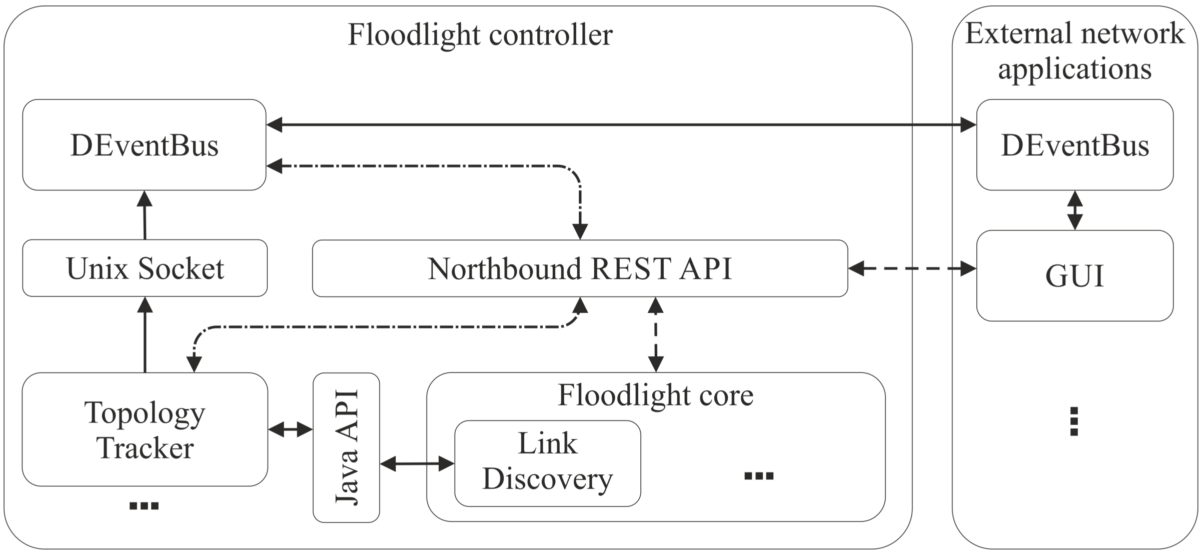


Рис. 2. Схема обмена сообщениями об изменениях сетевой топологии ПКС

3. Сравнение модулей Topology Tracker и Link Discovery

Анализ разработанного внешнего модуля Topology Tracker позволяет провести частичное сравнение по ключевым параметрам с аналогичным модулем Link Discovery, который в данный момент используются в контроллере Floodlight.

С точки зрения обеспечения безопасности контроллера. Обращение к внутреннему модулю контроллера Floodlight происходит по REST API запросу с любого устройства сети, при этом он может происходить как с указанием логина и пароля, так и без них. Взаимодействие с разработанным модулем могут осуществлять сетевые приложения, зарегистрированные в DEventBus. Ответственность за регистрацию сетевых приложений возложена на администратора сети.

С точки зрения используемых ресурсов контроллера. Разработанная система программных решений использует собственную базу данных, которая формируется в оперативной памяти по мере работы контроллера Floodlight параллельно с его собственной. Такой подход позволяет быстро формировать ответ на запрос о предоставлении полной или частичной информации о текущем состоянии сетевой топологии. Кроме того, разработанная структура имеет меньший размер, чем собственная база контроллера (расположенная *$floodlight_path$ /net/floodlightcontroller/storage*), так как содержит только информацию о хостах, сетевом оборудовании и связи между ними. Технически созданная база данных является java-коллекцией типа HashMap, что обеспечивает в лучшем случае константное ($O(1)$) время поиска, вставки и удаления элемента. В худшем случае эти операции выполняются за линейное время ($O(n)$). Так как HashMap в Java использует метод цепочек для разрешения коллизий, среднее время поиска элемента составляет $\Theta(1+\alpha)$, где α – коэффициент заполнения таблицы [20]. Стоит отметить, что собственная база данных контроллера Floodlight также располагается в оперативной памяти и имеет ряд вложенных структур типа HashMap, что в худшем случае дает сложность основных операций $\Theta(n^k)$, где k – уровень вложенности. Таким образом, при небольшой или простой топологии времена выполнения запросов будут сопоставимы, а при сложной или большой топологии очевидно преимущество разработанного программного решения.

С точки зрения загруженности каналов связи при условии, что контроллер и сетевые приложения находятся на различных физических устройствах. В данном случае при использовании разработанного механизма загрузка входящего канала на контроллер от сетевых приложений становится равна нулю.

Заключение

Качественным отличием разработанной системы является возможность своевременной доставки информации о произошедших в сетевой топологии изменениях от контроллера Floodlight к сетевым приложениям. Кроме того, высвобождаются сетевые и процессорные ресурсы, которые, использовались для обработки входящих REST-запросов. В качестве недостатка следует отметить узкую специализацию разработанной базы данных.

Разработанные алгоритмы и созданная на их основе система обмена сообщениями, является универсальным средством взаимодействия контроллера Floodlight и сетевых приложений. Она может быть использована для разработки иных сетевых приложений, для которых критичным фактором является своевременность поступления информации от контроллера Floodlight.

По описанным в статье алгоритмам были разработаны сетевые приложения *Topology Tracker* и *DEventBus* для контроллера Floodlight, а также были получены свидетельства о государственной регистрации программ для ЭВМ Российской Федерации *№2015618629* и *№2015660314* соответственно.

Список литературы / References

[1] V. Sokolov et al., “A network analytics system in the SDN”, SDN&NFV: The Next Generation of Computational Infrastructure: 2014 International Science and Technology Conference “Modern Networking Technologies (MoNeTec)” (Moscow, October 27–29, 2014), 160–162.

[2] N. McKeown et al., “OpenFlow: Enabling Innovation in Campus Networks”, *ACM SIGCOMM Computer Communication Review*, **38**:2 (2008), 69–74.

[3] *Open Networking Foundation*, https://www.opennetworking.org/.

[4] David Orenstein, “Application Programming Interface”, *Computerworld*, 2010, http://www.computerworld.com/article/2593623/app-development/application-programming-interface.html.

[5] Sally Johnson, “Do SDN northbound APIs need standards?”, SearchSDN (January, 2013), http://searchnetworking.techtarget.com/feature/Do-SDN-northbound-APIs-need-standards.

[6] Dijkstra E. W., “A note on two problems in connexion with graphs”, *Numer. Math*, **1**:1 (1959), 269–271.

[7] Ryan Wallner and Robert Cannistra, “An SDN Approach: Quality of Service using Big Switch’s Floodlight Open-source Controller”, Proceedings of the Asia-Pacific Advanced Network, **35** (2013), 14–19.

[8] C. Demetrescu, G.F. Italiano, “A new approach to dynamic all pairs shortest paths”, **51**:6 (2004), 968–992.

[9] *Transmission Control Protocol. DARPA Internet Program. Protocol Specification*, RFC793,September, 1981, www.rfc-editor.org.

[10] M. Nikitinskiy, I. Alekseev., “A stateless transport protocol in software defined networks”, SDN&NFV: The Next Generation of Computational Infrastructure: 2014 International Science and Technology Conference “Modern Networking Technologies (MoNeTec)” (Moscow, October 27–29, 2014), 108–113.

[11] M.A. Nikitinskiy and I.V. Alekseev, “Analyzing the Possibility of Applying Asymmetric Transport Protocols in Terms of Software Defined Networks”, *Automatic Control and Computer Sciences*, **49**:2 (2015), 94–102.

[12] *Floodlight SDN OpenFlow Controller*, https://github.com/floodlight/floodlight.

[13] Pautasso C., Wilde E., Alarcon R., *REST: Advanced Research Topics and Practical Applications*, Springer-Verlag New York, 2014, ISBN: 978-1-4614-9298-6.

[14] *IEEE 802.1AB (LLDP) Specification*, http://standards.ieee.org/getieee802/download/802.1AB-2005.pdf.

[15] M. Rose and K. McCloghrie, *Structure and Identification of Management Information for TCP/IP-based Internets*, RFC1155, May, 1990, www.rfc-editor.org.

[16] M. Rose and K. McCloghrie, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, RFC1213, March, 1991, www.rfc-editor.org.

[17] J. Case et al., *A Simple Network Management Protocol (SNMP)*, RFC1157, May, 1990, www.rfc-editor.org.

[18] *OpenFlow Switch Specification, Version 1.3.4*, March, 2014, OF switch v.1.3.4.

[19] Alekseev I. and Nikitinskiy M., “EvenetBus Module for Distributed OpenFlow Controllers”, Proceedings of the 17th Conference of Open Innovations Association FRUCT (Yaroslavl, Russia, 20-24 April 2015), 3–8.

[20] Thomas H. Cormen et al., *Introduction to Algorithms*, 3rd., MIT Press, 2009, ISBN: 0-262-03384-4.

DOI: 10.18255/1818-1015-2015-6-852-861

Development of Active External Network Topology Module for Floodlight SDN Controller

Noskov A. A., Nikitinskiy M. A., Alekseev I. V.

Received December 15, 2015

Traditional network architecture is inflexible and complicated. This observation has led to a paradigm shift towards software-defined networking (SDN), where network management level is separated from data forwarding level. This change was made possible by control plane transfer from the switching equipment to software modules that run on a dedicated server, called the controller (or network operating system), or network applications, that work with this controller. Methods of representation, storage and communication interfaces with network topology elements are the most important aspects of network operating systems available to SDN user because performance of some key controller modules is heavily dependent on internal representation of the network topology. Notably, firewall and routing modules are examples of such modules. This article describes the methods used for presentation and storage of network topologies, as well as interface to the corresponding Floodlight modules. An alternative algorithm has been suggested and developed for message exchange conveying network topology alterations between the controller and network applications. Proposed algorithm makes implementation of module alerting based on subscription to the relevant events. API for interaction between controller and network applications has been developed. This algorithm and API formed the base for Topology Tracker module capable to inform network applications about the changes that had occurred in the network topology and also stores compact representation of the network to speed up the interaction process.

Keywords: software-defined network, Floodlight controller, external module, service, SDN, network topology, Topology Tracker, DEventBus, Link Discovery

For citation: Noskov A. A., Nikitinskiy M. A., Alekseev I. V., "Development of Active External Network Topology Module for Floodlight SDN Controller", *Modeling and Analysis of Information Systems*, **22**:6 (2015), 852–861.

On the authors:
Noskov Andrey Aleksandrovich, orcid.org/0000-0002-2268-4912, engineer,
A-Real Group, Energiya-Info Inc., Souznaya str., 144, Yaroslavl, 150008, Russia, e-mail: naa@a-real.ru

Nikitinskiy Mikhail Aleksandrovich, orcid.org/0000-0001-8830-8613, system analyst, programmer,
A-Real Group, Energiya-Info Inc., Souznaya str., 144, Yaroslavl, 150008, Russia, e-mail: man@a-real.ru

Alekseev Igor Vadimovich, orcid.org/0000-0001-8321-2399, Director of the Internet Center, Ph.D.,
P.G. Demidov Yaroslavl State University, Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: aiv@yars.free.net

Acknowledgments:
This work was performed in the Yaroslavl State University with support of the Ministry of Education and Science of the Russian Federation in the framework of the Subsidy Agreement (*ID RFMEFI57414X0036*).