УДК 004.413

АВТОМАТИЗАЦИЯ ПРОВЕРКИ НЕКОРРЕКТНОСТИ КОНФИГУРИРОВАНИЯ СЕТЕВЫХ ЭКРАНОВ

В.В. Девятков, **Мьо Тан Тун**

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация
e-mail: deviatkov@bmstu.ru; myothanhtun@gmail.com

Рассмотрена автоматизация поиска ошибок конфигурирования межсетевых экранов (брандмауэров) на основе принципов логического анализа. Как правило, для описания поведения сетевых экранов использованы процессные модели, а для формулировки требований (свойств) некорректности — язык модальной логики. В настоящей статье детально изложена методика перехода от процессных моделей описания поведения сетевых экранов и условий их некорректности, формулируемых на языке модальной логики, к конкретным логическим программам, реализующим проверку некорректности конфигурирования сетевых экранов.

***Ключевые слова*:** межсетевой экран, процесс, модальная логика, язык логического программирования PROLOG.

AUTOMATION OF VERIFICATION OF INCORRECTNESS FOR FIREWALL CONFIGURATIONS

V.V. Devyatkov, **Myo Than Tun**

Bauman Moscow State Technical University, Moscow, Russian Federation
e-mail: deviatkov@bmstu.ru; myothanhtun@gmail.com

This article is devoted to the automation of the search of firewall configuration errors on the basis of logical analysis. As a rule, process models have been used for a description the behavior of firewalls, and the language of modal logic has been applied in order to formulate requirements (properties) of incorrectness. We focus on the presentation of a detailed methodology of the transition from process models of a description the behaviors of firewalls and conditions of their incorrectness (which were formulated in the language of modal logic) to the specific logical programs that allows to implement verification of incorrectness of firewall configurations.

***Keywords*:** firewall, process, modal logic, logic programming language PROLOG.

Введение. Сетевой экран, или брандмауэр, — широко известное средство защиты сетей. Для того чтобы обеспечить требуемую защиту, брандмауэр должен быть соответствующим образом настроен, или конфигурирован. К сожалению, поведение сетевого экрана может быть сопряжено с ошибками, которые допускают даже опытные администраторы, что приводит к снижению уровня защиты сети и проникновению в сеть нежелательных пакетов.

Для анализа корректности описания поведения сетевых экранов в работе [1] предложено использовать процессные модели для описания

поведения сетевых экранов, а для описания типовых ошибок поведения — язык модальной логики, на котором записываются наиболее типичные требования (их выполнение позволяет исключить определенные типы ошибок). Там же рассмотрены принципы создания логических программ проверки правильности некорректностей, приведены примеры логических программ проверки правильности поведения брандмауэров на языке VISUAL PROLOG.

Настоящая статья посвящена анализу корректности описания поведения сетевых экранов по тем же принципам, которые изложены в работе [1]. Однако в отличие от работы [1] здесь упор сделан на изложение детальной методики перехода от процессных моделей описания поведения сетевых экранов и условий их корректности, формулируемых на языке модальной логики, к конкретным логическим программам, реализующим проверку корректности конфигурирования сетевых экранов.

Сначала изложим принципы описания поведения сетевых экранов процессными моделями в виде, который наиболее естественно позволит перейти к представлению их в языке логического программирования VISUAL PROLOG.

Принципы анализа поведения процессных графов переходов на языке логического программирования VISUAL PROLOG. Традиционное описание поведения сетевых экранов осуществляется списками управления доступом. Каждый список управления доступом состоит из списка правил. Отдельное правило, также называемое условием перехода, представляется парами ?*a*.!*a* или ?not*a*.!*e*, где ?*a*, ?not*a* — восприятия, после получения которых сетевой экран совершает действия (реакции) !*a* или !*e*. Правила обрабатываются в определенном порядке, зависящем от вида списка управления доступом. Если имеем восприятие ?*a*, то выполняется реакция !*a* и переход к реализации очередного правила. Если имеем восприятие ?not*a*, то никакой реакции нет (реакция является пустой и обозначается символом !*e*) и начинается очередное правило.

В процессных графах переходов, описывающих поведение сетевых экранов, каждое состояние процесса изображается кружком, внутрь которого помещается символ этого состояния. Из каждого состояния возможен переход только в два других состояния. Переходы из одного состояния происходят в результате выполнения альтернативных правил ?*a*.!*a* или ?not*a*.!*e*. Начальное состояние обозначается двойным кружком, а финальное (если оно есть) — жирным кружком.

Переходы из состояния b_i в состояние b_j в результате выполнения правила ?*a*.!*a* представляется тройкой $(b_i, ?a.!a, b_j)$, а из состояния b_i в состояние b_j в результате выполнения правила ?not*a*.!*e* – тройкой $(b_i, ?\text{not}a.!e, b_j)$. Путем на графе, ведущем из состояния b_i в состояние

b_j, называется последовательность переходов, ведущих из состояния b_i в состояние b_j. Если из состояния b_i существует какой-либо путь в состояние b_j, то будем утверждать, что состояние b_j достижимо из состояния b_i этим путем. Путь, ведущий из состояния b_i в состояние b_j через состояния, среди которых не встречается ни одной пары одинаковых, называется простым. В противном случае путь называется сложным. Простой путь, ведущий из состояния b_i в то же самое состояние b_i, называется циклом. Сложный путь, ведущий из состояния b_i в то же самое состояние b_i, называется контуром. Состояние b_i, из которого ведет некоторый путь, называется начальным состоянием этого пути, а состояние b_j, в который этот путь ведет, — конечным состоянием данного пути.

Анализ корректности описания поведения сетевых экранов с помощью логических программ на языке логического программирования VISUAL PROLOG сводится к поиску свойств процессных графов переходов, наличие которых недопустимо с позиции корректности сетевых экранов. Например, это может быть наличие пар путей, которые ведут в состояния, вызывающие определенного рода противоречия.

В синтаксисе языка VISUAL PROLOG каждое состояние b_i представляется константой bi. Переменные, областью значений которых являются состояния, обозначаются прописными символами *B* и индексируются, если это необходимо, например, как Bi. Условия перехода ?*a*.!*a* или ?not*a*.!*e* представляются функторами, структура которых может быть достаточно сложной. Пока соответствующие функторы обозначим как *f*(?*a*.!*a*) или *f*(?not*a*.!*e*). С учетом принятых обозначений для введения структуры применяемых логических программ на языке VISUAL PROLOG запишем следующие предикаты и правила:

- *initial*(*Bi*) — одноместный предикат, истинный, когда автомат находится в начальном состоянии *Bi*;
- *final*(*Bj*) — одноместный предикат, истинный, когда автомат находится в конечном состоянии *Bj*;
- *transition*(*Bi,f*(?*a*.!*a*)*, Bj*), *transition*(*Bi, f*(?not*a*.!*e*)*, Bj*) — трехместные предикаты, называемые переходами, истинные, если на процессном графе переходов имеются переходы из состояния *Bi* в состояние *Bj*, условиями перехода которых являются ?*a*.!*a* или ?not*a*.!*e*;
- *accessible*(*Bi*, [*X*]*, Bj*)) — двуместный предикат, истинный, если состояние *Bj* достижимо из состояния *Bi* в результате последовательного выполнения условий перехода из представленных в этом списке слева направо (наличие на процессном графе переходов пути, ведущем из состояния *Bi* в состояние *Bj*, дуги которого помечены условиями перехода списка [*X*]);
- *accessible*(*Bi*, *X, Bj*):- *transition*(*Bi*, *X, Bj*) — нерекурсивное правило достижимости состояния *Bj* из состояния *Bi*;

- *accessible*(*Bi*, [*X*|*Rest*], *Bj*):- *transition*(*Bi*, *X, Bk*), *accessible*(*Bk*, [*Rest*], *Bj*) — рекурсивное правило достижимости состояния *Bj* из состояния *Bi*.

Рассмотрим процессный граф переходов, показанный на рис. 1. Здесь есть пара путей $(b_1, ?a_1.!a_1, b_2)$ и $(b_1, ?nota_1.!e, b_2)(b_2, ?nota_2.!e, b_3)(b_3, ?a_1.!a_3, b_4)$. Первый путь содержит один переход, на котором в ответ на восприятие $?a_1$ осуществляется реакция $!a_1$. Во втором пути последним переходом является переход $(b_3, ?a_1.!a_3, b_4)$, на котором в ответ на восприятие $?a_1$ выполняется реакция $!a_3$. Применительно к проверке корректности сетевых экранов (как это будет ясно далее) при определенной интерпретации наличие двух таких путей на процессном графе переходов недопустимо. Тогда логическая программа должна обнаруживать подобные пары путей.

Логическая схема программы (псевдокод) на языке VISUAL PROLOG 5.0 для процессного графа переходов графа, приведенного на рис. 1, позволяющая обнаружить наличие указанной пары путей, будет следующей:

```
domains
   f= f(symbol)
   list=f*
predicates
   nondeterm transition(symbol, symbol, symbol)
   nondeterm accessible(symbol, list, symbol)
clauses
   transition(b1, f(?a1,!a1), b2).
   transition(b1, f(?nota1,! !e), b2).
   transition(b2, f(?a2.!a2), b3).
   transition(b2, f(?nota2, !e), b3).
   transition(b3, f(?a1,!a3), b4).
```

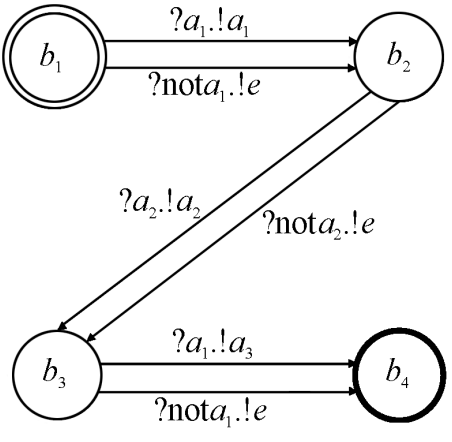

Рис. 1. Процессный граф переходов

transition(*b*3, *f*(?nota1, !*e*), *b*4).

accessible(*B*1, [*X*], *B*2) :- *transition*(*B*1, *X*, *B*2).

accessible(*B1*, [*X*|*Rest*], *B*2) :- *transition*(*B*1, *X*, *B*3),
accessible(*B*3, [*Rest*], *B*2).

goal

accessible(*b1*, [*f*(?*a*1,!*a*1)], *b*2),
accessible(*b1*, $[f(?nota_1, !e), f(?nota_2, !e), f(?a_1, !a_3)]$, *b*4).

Эта программа содержит четыре раздела: *domains*; *predicates*; *clauses*; *goal*. Раздел *domains* программы включает в себя описание структуры условий перехода, каждое из которых состоит из восприятия и реакции, являющихся символическими (*symbol*). В раздел *predicates* входит описание структуры используемых предикатов с предикатными символами *transition* и *accessible*, в раздел *clauses* — описание рассмотренных выше предикатов (фактов) и правил. Раздел *goal* ставит задачу нахождения двух путей, ведущих из состояния *b*1 в состояние *b*2 и из состояния *b*1 в состояние *b*4, содержащих при переходе в конечное состояние каждого пути различные непустые реакции на одно и то же восприятие.

Процессные графы переходов, адекватные списковым моделям. В работе [1] были изложены основные принципы перехода от спискового описания поведения сетевых экранов к описанию их поведения процессными графами переходов.

Пример процессного графа переходов, представляющего простую списковую модель, взятую из работы [2], показан на рис. 2, *а*. На этом графе символы *p* обозначают пакеты ip-адресов и другой сопутствующей информации. Пример процессного графа переходов, адекватного сложной списковой модели, представлен на рис. 2, *б*.

В типичной сетевой среде несколько экранов чаще всего распределяются по сети и конфигурируются независимо друг от друга. В этом случае защищенность сети зависит от правильности настройки всех сетевых экранов так, чтобы поведение одних сетевых экранов не понижало уровень защищенности, обеспечиваемый другими экранами. Например, если в какую-либо защищаемую зону информация может поступать из двух разных источников через различные сетевые экраны $П_1$ и $П_2$, а каждый сетевой экран описывается процессными графами переходов P_1 и P_2, адекватных своим списковым моделям, то проверка правильности настройки сетевых экранов сводится к проверке свойств параллельного поведения процессных графов [3–9].

Формулировка некорректностей поведения процессных графов переходов, адекватных списковым моделям. При рассмотрении некорректностей в работе [1] была использована классификация

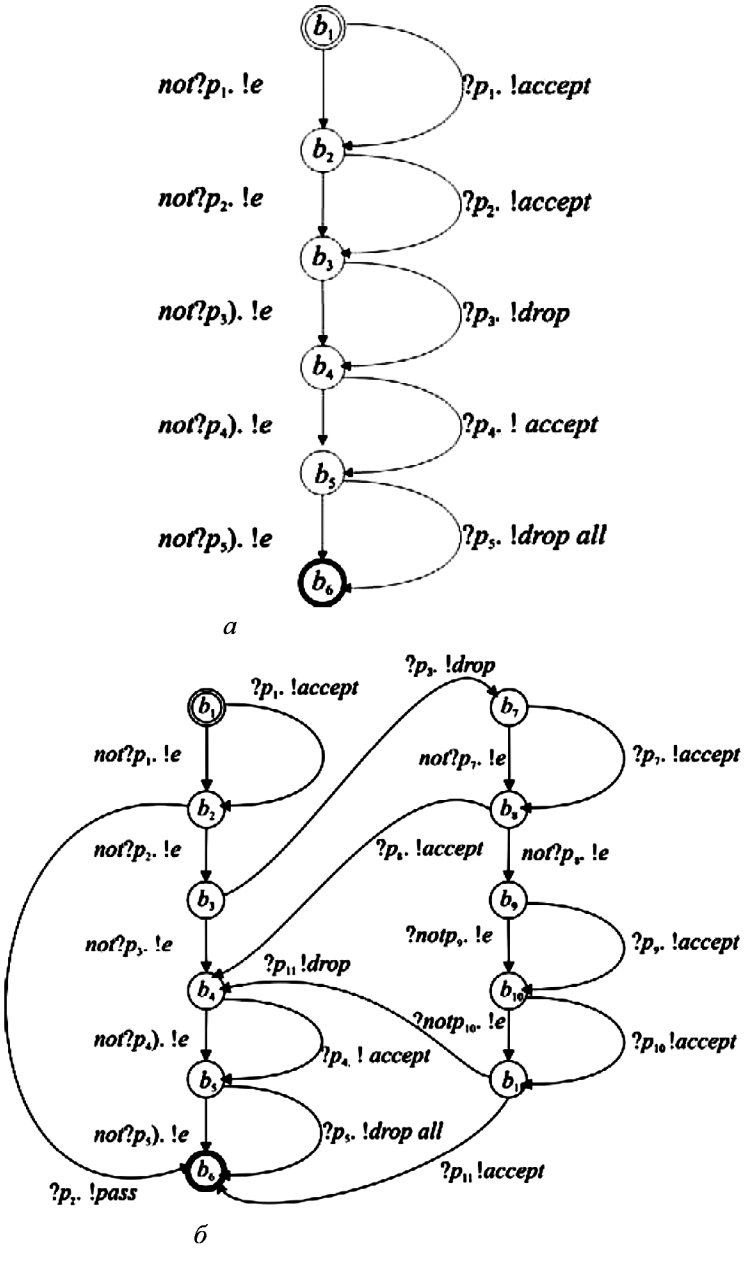

Рис. 2. Процессные графы переходов простой (*а*) и сложной (*б*) списковых моделей

некорректностей описания списковых моделей, приведенная в работе [2]. В работе [1] для формулировки условий некорректного описания этих моделей предложено сначала представить их в виде процессных графов переходов, далее сформулировать условия корректности поведения этих графов как высказываний на языке временно́й модальной логики, а затем получить условия некорректного описания

как логического отрицания условий корректности. Как уже было отмечено, в настоящей статье изложены принципы и процедура перехода от процессных графов переходов и полученных указанным образом условий некорректности к логическим программам на языке VISUAL PROLOG, позволяющим осуществлять проверку некорректностей. Такую процедуру перехода рассмотрим на примере единственной некорректности, называемой затенением. Для остальных некорректностей процедуры перехода, за исключением несущественных отличий, аналогичны.

В работе [1] на языка модальной логики простейшее условие (правило), выполнение которого гарантирует отсутствие затенения пакета p, имеет вид

$$\Box[(?p\wedge!accept) \supset \neg\Diamond(?p\wedge!drop)].$$

Смысл этого правила следующий: всегда, когда в каком-либо текущем состоянии осуществляется восприятие пакета p, над которым совершается действие $\underline{!}accept$, тогда не должно быть достижимого из этого состояния другого состояния, в которое процесс переходит в результате восприятия того же пакета p, над которым совершается действие $drop$. Наличие хотя бы одного затенения означает истинность формулы

$$\begin{gathered}\neg(\Box[(?p\wedge!accept) \supset \neg\Diamond(?p\wedge!drop)]) \equiv\\ \Diamond\neg[(?p\wedge!accept) \supset \neg\Diamond(?p\wedge!drop)] \equiv\\ \Diamond[\neg((?p\wedge!accept) \supset \neg\Diamond(?p\wedge!drop))] \equiv\\ \Diamond[\neg(\neg(?p\wedge!accept) \vee \neg\Diamond(?p\wedge!drop))] \equiv\\ \Diamond[(?p\wedge!accept) \wedge \Diamond(?p\wedge!drop)],\end{gathered}$$

т.е. в некотором текущем состоянии возможно восприятие пакета p, над которым совершается действие $!accept$, и в достижимом из этого состояния другом состоянии, в которое процесс переходит в результате восприятия того же пакета p, совершается действие $drop$.

По отношению к процессному графу это означает следующее: имеются два состояния $b2$ и $b3$, достижимые из начального состояния $b1$, одно из которых, например $b3$, достижимо из состояния $b2$ и при этом существуют переходы $(b_2, ?p.!accept, b_i)$, $(b_3, ?p.!drop, b_j)$. Такая ситуация является, скорей всего, ошибкой, поскольку непонятно, чем руководствовался автор процессного графа, указав, что на пути из начального состояния в состояние $b3$ сначала можно попасть в состояние $b2$, при переходе из которого разрешено принять пакет p, а затем попасть в состояние $b3$, при переходе из которого этот пакет принимать запрещено.

Рассмотренное условие некорректности $\Diamond[(?p\wedge!accept) \wedge \Diamond(?p\wedge!drop)]$ сформулировано для одного и того же пакета. Именно поэтому оно названо простейшим. На самом деле затенения могут быть более сложными. Например, несколько пакетов могут затенять

один, пакеты могут не совпадать, пересекаться, быть подмножествами один другого и т.п.

Язык временно́й модальной логики не использует в явном виде состояния и правила вывода. В рассматриваемом случае исходными для анализа некорректностей сетевых экранов, во-первых, являются процессные графы переходов, содержащие состояния в явном виде, во-вторых, вывод осуществляется в языке логического программирования с присущей ему стратегией вывода. При переходе от описания некорректностей на язык логического программирования VISUAL PROLOG состояния будут появляться в явном виде в качестве термов в фактах и правилах.

На примере простейшего затенения рассмотрим формирование на языке VISUAL PROLOG конкретной логической программы проверки этой некорректности.

Процедура формирования логической программы проверки некорректности. Используем процессный граф переходов, приведенный на рис. 2, *а*. Предположим, что пакеты, именованные на этом графе, являются следующими:

p_1 = *udp 192.168.1.1,*

p_2 = *tcp 10.1.1.0,*

p_3= *udp 192.168.1.1,*

p_4= *tcp 10.1.1.26,*

p_5= *tcp 10.1.1.64.*

Пакет p_1 затеняется пакетом p_3, поскольку они совпадают, но согласно процессному графу (см. рис. 2, *а*) первый пакет должен быть принят (*accept*), а второй — запрещен (*drop*).

В зависимости от структуры описания пакетов в разделе *domains* могут описываться различные типы функторов. В данном примере в разделе *domains* объявим функтор *f*(*symbol, integer, integer, integer*, *integer*, *symbol*), элементами которого являются тип протокола, ip-адрес, состоящий из четырех целых чисел, и действие, совершаемое над ним. Здесь же введем список, содержащий описанные функторы. В результате имеем

domains

f =*f*(*symbol, integer, integer, integer*, *integer*, *symbol*)

list = *f**

Раздел *predicates* включает в себя описание тех же предикатов, что и логическая схема программы, приведенная ранее:

predicates

nondeterm transition(*symbol, f , symbol*)

nondeterm accessible(*symbol, list, symbol*)

В разделе *clauses* логической программы каждому переходу (b_i,?*a*.!*a*, b_j) на процессном графе ?*a*= *udp 192.168.1.1*, !*a* = !*accept*

соответствует факт *transition*(*bi*, *f*(*udp,192,168,11, accept*), *bj*), а переходу $(b_i, ?nota.!e, b_j)$ — факт *transition*(*bi*, *f*(*udp,192,168,11, e*), *bj*). Правила остаются такими же, как и в схеме программы, приведенной выше. Таким образом, для примера запишем следующий раздел *clauses*:

clauses

transition(*b*1, *f*(*udp,192,168,1,1, accept*), *b*2).
transition(*b*2, *f*(*tcp,10,1,1,0, accept*), *b*3).
transition(*b*3, *f*(*udp,192,168,1,1,drop*), *b*4).
transition(*b*4, *f*(*tcp,10,1,1,26, accept*), *b*5).
transition(*b*5, *f*(*tcp,10,1,1,64, drop all*), *b*6).
transition(*b*1, *f*(*udp,192,168,1,1,e*), *b*2).
transition(*b*2, *f*(*tcp,10,1,1,0,e*),*b*3).
transition(*b*3, *f*(*udp,192,168,1,1,e*),*b*4).
transition(*b*4, *f*(*tcp,10,1,1,26,e*),*b*5).
transition(*b*5, *f*(*tcp,10,1,1,64, e*),*b*6).

accessible(*B*1, [*X*], *B*2) :- *transition*(*B*1, *X, B*2).
accessible(*B*1, [*X*|*Rest*], *B*2) :- *transition*(*B*1, *X, B*3), *accessible*(*B*3, *Rest, B*2).

В разделе *goal* формулируются такие цели, как задача нахождения путей. Для рассматриваемого примера на языке VISUAL PROLOG одну из целей можно выразить, например, в следующем виде:

goal

accessible(*b*1,[*f*(*A,B,C,D,E,accept*)] ,*B*1),*accessible*(*b*1,[*f*(*_,_,_,_,_,_*), *f*(*_,_,_,_,_,_*),*f*(*A,B,C,D,E,drop*)],*B*2).

Здесь необходимо узнать, сколько существует пар путей длиной 1 и 3, ведущих из состояния *b*1 в состояния *B*1 и *B*2 и каковы эти состояния, при переходе в которые одинаковые пакеты p_1 = *udp 192.168.1.1* и *p*3= *udp 192.168.1.1* в первом случае принимаются, а во втором — запрещаются.

В результате выполнения описанной логической программы на языке VISUAL PROLOG получаем четыре решения:

*A=udp, B=192, C=168, D=1, E=1, B*1=*b*2, *B*2=*b*4
*A=udp, B=192, C=168, D=1, E=1, B*1=*b*2, *B*2=*b*4
*A=udp, B=192, C=168, D=1, E=1, B*1=*b*2, *B*2=*b*4
*A=udp, B=192, C=168, D=1, E=1, B*1=*b*2, *B*2=*b*4
4 Solutions

Каждое решение означает наличие своей, отличной от других, пары путей, ведущих в состояния *b*2 и *b*4, приводящих к затенению.

Заключение. В настоящей статье подробно изложена методика перехода от конфигурирования сетевых экранов процессными моделями и условий их некорректности к логической программе на языке

VISUAL PROLOG. Методика проиллюстрирована простым примером проверки некорректности, называемой затенением. Планируется расширение указанной методики на другие виды некорректностей, а также теоретическое и экспериментальное обоснование ее трудоемкости логическими программами.

ЛИТЕРАТУРА

1. *Девятков В.В.*, *Мьо Тан Тун*. Логический анализ корректности конфигурирования межсетевых экранов // Инженерный журнал: Математическое моделирование. 2013. Вып. 11. URL: http://www.engjournal.ru/catalog/it/security/988.html
2. *Lihua Yuan*, *Jianning Mai*, *Chen-Nee Chuah*. FIREMAN: A Toolkit for FIREwall Modeling and Analysis. URL: http://www.cse.psu.edu/ mcdaniel/cse544/slides/cse544-fireman-lin.pdf
3. *Девятков В.В.* Системы искусственного интеллекта. М.: Изд-во МГТУ им. Н.Э. Баумана, 2001. 352 с.
4. *Firewall* wizards security mailing list. [Электронный ресурс] URL: http://honor.icsalabs.com/mailman/listinfo/firewall-wizards
5. *Девятков В.В.* Построение, оптимизация и модификация процессов // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. 2012. № 2. С. 60–79.
6. *Wool A.* A quantitative study of firewall configuration errors // IEEE Computer. Vol. 37 (6). 2004. P. 62–67. DOI: 10.1109/MC.2004.2
7. *Manna Z.*, *Pnueli A.* Temporal verification of reactive systems: progress. N.Y.: Draft, 1996.
8. *Леммон Е.* Алгебраическая семантика для модальных логик I. II. В кн.: Семантика модальных и интенсиональных логик. М.: Процесс, 1981. 424 с.
9. *Основы* программирования на языке Пролог: Информация. URL: http://www.intuit.ru/studies/courses/44/44/info
10. *Основы* программирования на языке Visual Prolog. URL: http://www.intuit.ru/studies/courses/12333/1180/info

REFERENCES

[1] Devyatkov V.V., Tun M. Tan. Logical analysis of the correctness of the of firewall configurations. *Jelektr. nauchno-tehn. Izd. "Inzhenernyy zhurnal: nauka i innovacii" MGTU im. N.E. Baumana: Informatsionnye tekhnologii* [El. Sc.-Tech. Publ. "Eng. J.: Science and Innovation" of Bauman MSTU: Information technology], 2013, iss. 11 (23), 17 p. (in Russ.). Available at: http://www.engjournal.ru/catalog/it/security/988.html (accessed 24.11.2014).

[2] Yuan L., Mai J., Su Zh., Chuah Ch.-N., Chen H., Su Zh. FIREMAN: A toolkit for firewall modeling and analysis. *Proc. of the IEEE Symposium on Security and Privacy* 2006, pp. 199–213. Available at: http://www.cse.psu.edu/ mcdaniel/cse544/slides/cse544-fireman-lin.pdf (accessed 24.09.2014).

[3] Devyatkov V.V. Sistemy iskusstvennogo intellekta [Artificial intelligence systems]. Moscow, MGTU im. N.E. Baumana Publ., 2001. 352 p.

[4] Firewall wizards security mailing list. Available at: http://honor.icsalabs.com/mailman/listinfo/firewall-wizards (accessed 24.09.2014).

[5] Devyatkov V.V. Building, optimization and modification of processes. *Vestn. Mosk. Gos. Tekh. Univ. im. N.E. Baumana, Priborostr.* [Herald of the Bauman Moscow State Tech. Univ., Instrum. Eng.], 2012, no. 2, pp. 60–79 (in Russ.).

[6] Wool A. A quantitative study of firewall configuration errors. *IEEE Computer*, 2004, vol. 37, no. 6, pp. 62–67. DOI:10.1109/MC.2004.2

[7] Manna Z., Pnueli A. Temporal verification of reactive systems: progress. N.Y., Springer-Verlag, 1996. Draft manuscript.

[8] Lemmon E. Algebraicheskaya semantika dlya modal'nykh logik I. II. V kn.: Semantika modal'nykh i intensional'nykh logik. Pod red. Smirnova V.A. [Algebraic semantics for modal logics I. II. In book: The semantics of modal and intensional logics. Ed. by Smirnov V.A.]. Moscow, Progress Publ., 1981. 424 p.

[9] Osnovy programmirovaniya na yazyke Prolog: Informatsiya [Fundamentals of Programming in Prolog: Information]. Available at: http://www.intuit.ru/studies/courses/44/44/info (accessed 24.09.2014).

[10] Osnovy programmirovaniya na yazyke Visual Prolog [Fundamentals of Programming in Visual Prolog]. Available at: http://www.intuit.ru/studies/courses/12333/1180/info (accessed 24.09.2014).

Статья поступила в редакцию 24.09.2014

Девятков Владимир Валентинович — д-р техн. наук, профессор, заведующий кафедрой "Информационные системы и телекоммуникации" МГТУ им. Н.Э. Баумана. Автор более 120 научных работ (в том числе трех монографий) в области искусственного интеллекта, распознавания образов, принятия решений, логических исчислений, представления знаний, теории конечных автоматов, логического синтеза и анализа дискретных устройств и систем.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.

Devyatkov V.V. — Dr. Sci. (Eng.), professor, head of "Information System and Telecommunication" department of the Bauman Moscow State Technical University. Author of more than 120 publications (including three monographs) in the field of artificial intelligence, pattern recognition, decision making, logic calculi, knowledge representation, theory of finite state machines, logical synthesis and analysis of discrete devices and systems.
Bauman Moscow State Technical University, 2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.

Мьо Тан Тун — аспирант кафедры "Информационные системы и телекоммуникации" МГТУ им. Н.Э. Баумана.
МГТУ им. Н.Э. Баумана, Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5.

Myo Than Tun — post-graduate of "Information System and Telecommunication" department of the Bauman Moscow State Technical University.
Bauman Moscow State Technical University, 2-ya Baumanskaya ul. 5, Moscow, 105005 Russian Federation.