УДК 004.4'233

**Д.А. Эдель**

## Языковая модель исполнимых кодов

ПО аналогии с методами сравнения естественных языков строится языковая модель исполнимого кода на основе однородных Марковских цепей и приводятся результаты экспериментальных исследований предложенной модели.
**Ключевые слова:** машинный код, Марковские цепи, классификация.

**Введение.** В настоящее время существует множество задач, при решении которых необходимо применять методы анализа и трансформации программ. Это задачи обеспечения безопасности программ, генерации и верификации исходных кодов, а также обратной инженерии. С целью решения таких задач используются дизассемблеры и декомпиляторы, при работе которых встает целый ряд подзадач, уже имеющих решения на текущий момент. Однако существуют и задачи, требующие более эффективных решений, чем существующие. Одна из таких задач – разделение неопределенных участков программ на код и данные.

В работе [1] описывается результативность существующих методов разделения таких участков, суть которых сводится к сигнатурному анализу, не способному полностью решать поставленную проблему. В данной статье проверяется гипотеза, что однородные Марковские модели первого порядка смогут эффективно моделировать исполнимый код для задачи разделения неопределенных участков программ на код и данные, и исследуется эффективность метода на практике.

**Основной текст.** Любую последовательность байт длины более 16, согласно спецификации Intel IA-32 [2], можно дизассемблировать как последовательность команд, причем единственным образом. Если дизассемблирование производить с разных позиций, то появится эффект наложения команд, детально описанный в статье [3] , при котором один и тот же байт может входить в несколько исполнимых команд, как показано в табл. 1.

Таблица 1

**Эффект наложения команд**

<table>
<tr><td rowspan="11">Двоичное представление кода</td><td>14<br>53</td><td>Последовательность 1</td><td>Последовательность 2</td><td>Последовательность 3</td></tr>
<tr><td rowspan="2">add esp, 14</td><td></td><td rowspan="2"></td></tr>
<tr><td>push ebx</td></tr>
<tr><td>83<br>ec<br>0c</td><td>sub esp, c</td><td>sub esp, c</td><td rowspan="2">or al, 8b</td></tr>
<tr><td rowspan="3">8b<br>5c<br>24<br>14</td><td rowspan="3">mov ebx, 14</td><td rowspan="3">mov ebx, 14</td></tr>
<tr><td>pop esp</td></tr>
<tr><td>and al, 14</td></tr>
<tr><td>8b<br>53<br>04</td><td>mov edx, 4</td><td>mov edx, 4</td><td>mov edx, 4</td></tr>
<tr><td>8b<br>0b</td><td>mov ecx, ebx</td><td>mov ecx, ebx</td><td>mov ecx, ebx</td></tr>
<tr><td colspan="4">Дизассемблированные иструкции</td></tr>
</table>

Следовательно, одна и та же последовательность байт может формировать различные последовательности исполнимых команд в зависимости от позиции начала дизассемблирования. Указанный факт является основной проблемой в задаче разделения неопределенных участков программ на код и данные, т.к. любой фрагмент данных можно интерпретировать как код, хотя он таковым не является, и при его исполнении на центральном процессоре произойдет ошибка выполнения.

Так как код программы представляет собой логически завершенную последовательность команд, реализующую конкретный алгоритм на языке высокого уровня, то порядок команд строго фиксирован в цепочке и логически упорядочен. Указанная семантическая особенность команд исполнимого кода является их специфическим отличием от команд, полученных при дизассемблировании случайного фрагмента данных.

Аналогично в естественных языках слова в предложениях, как правило, имеют упорядоченность на основе семантики и синтаксиса языка. Некоторые комбинации слов и букв соответствуют различным языкам.

Пример корректной исполнимой последовательности команд языка Ассемблер представлен в табл. 2.

Таблица 2

**Пример корректной исполнимой последовательности команд**

| Байт код команд | Команда | Значение |
|---|---|---|
| **55**<br>**8B** EC<br>**83** EC 18 | PUSH EBP<br>MOV EBP, ESP<br>SUB ESP, 18 | Стандартный код начала функции |
| **6A** 00<br>**6A** 00<br>**6A** 00<br>**E8** 23 24 FEFF | PUSH 0<br>PUSH 0<br>PUSH 0<br>CALL 00445606 | Передача параметров<br><br><br>Вызов функции |
| **85** C0<br>**0F 84** F0 9D FD FF | TEST EAX, EAX<br>JESHORT 0043CFDB | Проверка условия<br>Условный переход |

В приведенной таблице показано, что первые три команды, как правило, представляют начало функции, три «push» и «call» представляют собой вызов функции с передачей ей параметров, а «test» и «je» представляют собой проверку условия и условный переход. Жирным выделены значения в байтовой последовательности, отвечающие за сами команды, оставшиеся байты – параметры команд.

В лингвистике одним из наиболее успешных методов определения языка по тексту (из конечного числа языков) являются Марковские модели [4]. В данной работе на основе аналогии с методами сравнения естественных языков проверяется *гипотеза, что однородные Марковские модели первого порядка смогут эффективно моделировать исполнимый код в задаче разделения неопределенных участков программ на код и данные.*

За последовательность дискретных случайных величин $\{X_n\}$, $n \geq 0$ примем последовательность команд (без параметров), в которой

$$P(X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = P(X_{n+1} = i_{n+1} | X_n = i_n), \qquad (1)$$

т.е. на вероятность появления следующей команды в цепочке влияет только текущая команда. Тогда $\{X_n\}$ образует однородную Марковскую цепь первого порядка, в которой n – номер команды в последовательности. На начальном этапе проверки гипотезы матрицу переходных вероятностей $P_{ij}(n) = P(X_{n+1} = j | X_n = i)$ построим на основе серии экспериментов. Тогда, приняв за $\{O_j\}, j = 0 \ldots L$ – тестируемую последовательность команд, которую необходимо классифицировать, вероятность принадлежности ее определенной модели M будет выражаться по формуле

$$P_M(O) = P_M(O_1) \times P_M(O_2|O_1) \times P_M(O_3|O_2) \times \ldots \times P_M(O_L|O_{L-1}), \qquad (2)$$

Отметим, что если $\{O_j\}$ представляет собой корректную исполнимую последовательность команд длиной $L$, то существует только одна последовательность байт $\{O_j\}$, представляющая эту последовательность.

Основываясь на том факте, что в теле программы код функции может начинаться в любой позиции файла, можно получить несколько его интерпретаций в виде последовательностей команд. Различие в интерпретации будет определяться позицией дизассемблирования первой команды последовательности. В работе [3] изучен и доказан тот факт, что дизассемблированные последовательности команд, начатые с разных позиций файла (как показано в табл. 1), сойдутся с большой вероятностью менее чем через 32 байта в одну последовательность. Следовательно, для любого фрагмента программы более 64 байт можно получить последовательность команд$\{O_j\}$ длиной $L$, из которых первые $K(< 32 < L)$ будут командами «схождения» (различными командами на одинаковых позициях последовательностей), а оставшиеся $L - K$ команд будут истинными командами последовательности при дизассемблировании. Следовательно, при полном дизассемблировании фрагмента длиной более 64 байт можно с большой вероятностью получить истинную последовательность команд, которую она представляет, либо построить ложную последовательность, не характерную для исполнимых файлов.

**Экспериментальные исследования.** Построим модели исполнимых и неисполнимых последовательностей команд. Матрицы переходных вероятностей моделей изначально предлагается строить обучением на основе выборок файлов. Команды обоих

видов моделей на первом шаге примем равновероятными. Обучаемую выборку файлов (для построения матриц переходных вероятностей) будут представлять набор из исполнимых файлов формата Windows PE и различные файлы неисполнимых форматов. Согласно проводимому в работе сравнению естественных языков и машинного, по аналогии с разбиением текстов естественных языков на корпуса [4] , разделим исполнимые и неисполнимые файлы обучаемой выборки на классы и для каждого класса на основе описанного ниже алгоритма построим языковую модель, представляющую данный класс.

Для построения матрицы переходных вероятностей модели исполнимых файлов необходимо получить последовательности команд исполнимых файлов. Для этого необходимо построить граф потока управления и команд, начиная с точки входа в программу, и выделить все пути как корректные последовательности команд для обучения модели исполнимых файлов.

Для проверки, какой модели соответствует тестируемый файл (участок программы), предлагается следующий **алгоритм:**

1. Для каждой позиции файла строится дизассемблированная команда, начатая в этой позиции (на основе спецификации Intel IA-32 [2]).

2. Для каждой позиции файла строится последовательность команд, которая обрывается на позиции, с которой невозможно получить корректную команду.

3. На основе последовательностей, найденных в п. 2, формируются все возможные их подпоследовательности команд $S_j$ длины $T$.

4. Для каждой подпоследовательности команд $S_j$ по формуле (2) вычисляются вероятности соответствия исполнимым классам моделей. Среди всех подпоследовательностей выбирается одна $S'$, с самой большой вероятностью для любой модели исполнимого класса.

5. Найденная подпоследовательность $S'$, имеющая максимальную вероятность среди исполнимых моделей на файл (наилучший кандидат на исполнимую последовательность команд), аналогично п. 4 проверяется на соответствие оставшимся моделям неисполнимых классов.

6. Подпоследовательность команд $S'$ будем считать исполнимой, если она наиболее вероятна какому-нибудь классу исполнимой модели, иначе она считается неисполнимой.

7. Тестируемый фрагмент файла, участок программы будем считать исполнимым, если S' была принята за исполнимую подпоследовательность, иначе – не исполнимым.

Описанный алгоритм позволяет классифицировать неопределенные участки программ на код и данные. Минимальная длина подпоследовательности команд, которая используется при классификации, определяется экспериментально.

**Подготовка выборки файлов для обучения моделей.** Для формирования моделей было отобрано 23126 исполнимых и 62883 неисполнимых файлов. С помощью средства PeID (сигнатурный детектор компиляторов и упаковщиков программ [5] ) вся выборка исполнимых файлов была разделена на 13 классов. Неисполнимая выборка была разделена по формату файлов на 3 класса. Тестируемую выборку представлял набор из 30 файлов, не входящих в обучаемую: 15 исполнимых и 15 неисполнимых форматов. В моделях было 342 уникальные команды.

**Условия эксперимента.** Для проведения экспериментов использовался алгоритм, описанный выше, с тем изменением, что сначала длина $T$ находилась экспериментально. Алгоритм прекращал работу, как только все файлы обучаемой выборки успешно классифицируются. Было установлено, что при $T > 30$ обучаемая выборка успешно классифицируется.

При построении матриц переходных вероятностей моделей на основе встречаемости команд в подпоследовательностях возможен случай, когда паросочетание команд встречается в одной модели и отсутствует в другой. Согласно свойствам матриц переходных вероятностей, $\sum_{j=1}^{H} P_{ij}(n) = 1, \forall n \in N$ для каждого $j$, где H – общее количество уникальных команд. После нахождения частот встречаемости паросочетаний команд, каждая модель дополняется нулями для соответствующих пар, далее матрица переходных вероятностей нормируется и приводится к стохастическому виду. При нормировании используется формула

$$P'_{ij}(n) = P_{ij}(n) \times K + 1, \tag{3}$$

где $K = 2 \times (M - 1)$ – коэффициент нормирования; M – количество различных уникальных команд в цепочках.

В табл. 3 представлена нормированная и приведенная к стохастическому виду матрица переходных вероятностей $P_{ij}(n)$ для Марковской модели последовательности команд из табл. 2.

Таблица 3

**Стохастическая матрица переходных вероятностей**

| | call | je | mov | push | sub | test |
|---|---|---|---|---|---|---|
| call | 0,063 | 0,063 | 0,063 | 0,063 | 0,063 | 0,063 |
| je | 0,167 | 0,167 | 0,167 | 0,167 | 0,167 | 0,167 |
| mov | 0,063 | 0,063 | 0,063 | 0,063 | 0,688 | 0,063 |
| push | 0,239 | 0,022 | 0,239 | 0,457 | 0,022 | 0,063 |
| sub | 0,063 | 0,063 | 0,063 | 0,688 | 0,063 | 0,063 |
| test | 0,063 | 0,688 | 0,063 | 0,063 | 0,063 | 0,063 |

Нормирование матриц является необходимым действием при построении моделей, иначе вероятность подпоследовательности, вычисляемая по формуле (2), станет равной 0 как только встретится одно из паросочетаний команд, отсутствующее в модели.

$$P(O) = P(mov|push) \times P(sub|mov) \times P(push|sub) \times P(push|push) \times P(push|push) \times P(call|push) \times \\ \times P(test|call) \times P(je|test) = 0,239 \times 0,688 \times 0,688 \times 0,457 \times 0,457 \times 0,239 \times 0,688 \times 0,688 = 0,003. \quad (4)$$

В табл. 4 приведены значения вероятностей соответствия построенной модели различным последовательностям команд.

Таблица 4

**Вероятности соответствия различных последовательностей**

| | Последовательности команд | | | | | | | | | P |
|---|---|---|---|---|---|---|---|---|---|---|
| Команды | je | je | push | mov | mov | call | je | sub | call | 2E-08 |
| $P_{ij}(n)$ | 0,167 | 0,167 | 0,239 | 0,063 | 0,063 | 0,063 | 0,167 | 0,063 | - | |
| Команды | je | push | je | mov | je | call | je | je | call | 1E-08 |
| $P_{ij}(n)$ | 0,167 | 0,022 | 0,167 | 0,063 | 0,167 | 0,063 | 0,167 | 0,167 | - | |

На основе приведенного примера в табл. 2 можно сделать вывод о том, что модель, построенная по алгоритму, способна классифицировать неопределенные участки программ на код и данные. С целью проверки данного вывода и модели была проведена серия экспериментов.

**Результаты.** В табл. 5 представлен результат экспериментов.

Таблица 5

**Результаты экспериментов**

| Файл | Наилучшая модель | Вероятность | Значение T (длина последовательности) | Последовательность команд |
|---|---|---|---|---|
| 1.doc | DOCS_old | 0,15455 | 30 ($T_{max}$) | add - add - ...- add |
| 2.doc | DOCS_old | 0,00034 | 8 | add - add - add - add- add- add - push - push |
| 3.doc | DOCS_old | 0,87917 | 3 ($T_{min}$) | add - add - add |
| ... | ... | ... | ... | ... |
| 15.doc | DOCS_old | 0,82434 | 4 | add - add - add - add |
| 1.exe | EXEs | 0,99199 | 2 ($T'_{max}$) | leave - retn |
| 2.exe | EXEs | 0,99199 | 2 ($T'_{min}$) | leave - retn |
| ... | ... | ... | ... | ... |
| 15.exe | EXEs | 0,94009 | 2 | pusha - mov |

В представленной таблице $T_{min}$ – наименьшая длина подцепочки, при которой хотя бы один файл тестируемой выборки успешно классифицировался. Значение $T_{max}$ представляет собой минимальное $T$, при котором все файлы тестируемой выборки успешно классифицировались. Таблица отражает тот факт, что у исполнимых файлов встречаются паросочетания команд, вероятность которых максимальна для исполнимых классов моделей и минимальна для неисполнимых. В то же время существуют и такие комбинации команд, при которых неисполнимый файл на длинных последовательностях

Таблица 6

**Результат работы алгоритма для файла F**

| Шаг | Детектированный вид модели | Вероятность | Модель | T, длина цепочки | Позиция в файле | Последовательность команд |
|---|---|---|---|---|---|---|
| 1 | EXE | 0,002923977 | EXEs | 2 | 540 | cdwe - mov |
| 2 | EXE | 0,151741296 | Borland Delphi | 2 | 540 | cwde - mov |
| ... | ... | ... | ... | ... | ... | ... |
| 13 | EXE | 0,894492567 | Macromedia Windows Flash | 2 | 2 | adc - mov |
| 14 | EXE | 0,000175179 | MEW | 3 | 540 | cwde - mov - add |
| 15 | EXE | 0,004765652 | Borland Delphi | 3 | 540 | cwde - mov - add |
| ... | ... | ... | ... | ... | ... | ... |
| 26 | DOC | 0,879173038 | DOCs_old | 3 | 23 | add - add - add |
| Результат | DOC | 0,879173038 | DOCs_old | 3 | 23 | add - add - add |

классифицируется как исполнимый. В табл. 6 представлен ход работы алгоритма на файле «3.doc» из табл. 5.

В приведенной таблице показано, что согласно алгоритму поиска подпоследовательности команд с максимальной вероятностью находятся случайные команды в файле, исполнимое представление которых в виде команд дает большие значения на исполнимых моделях, что при очень больших размерах файла может давать неверные результаты.

**Выводы.** Алгоритм классифицирования неопределенных участков программ, основанный на основе однородных Марковских цепей и построенный как языковая модель исполнимого кода, успешно работает в решении задачи гарантированного выделения только исполнимых команд из общей выборки. Но в общем случае при решении задачи разделения неопределенных участков программ его необходимо усилить более качественной исходной выборкой данных, а также выделить из всего набора исполнимых команд наиболее значимые для построения модели каждого класса.

## *Литература*


1. *C. Linn S. D.* Obfuscation of Executable Code to Improve Resistance to Static Disassembly [Электронный ресурс]. — ACM Portal – The Guide to computing literature. — URL: http://portal.acm.org/%20citation.cfm?id=948149 ; (дата обращения: 29.04.2010).
2. IA-32 Assembly Language Reference Manual [Электронный ресурс]. — URL: http://docs.sun.com/app/docs/doc/806-3773 ; (дата обращения: 29.04.2010).
3. *N. Rosenblum X. Z., Miller B., Hunt K.* Machine Learning-Assisted Binary Code Analysis [Электронный ресурс]. — NIPS 2007 Workshop on Machine Learning in Adversarial Environments for Computer Security. — URL: http://mlsnips07.first.fraunhofer.de/abstracts/16-Rosenblum.pdf ; (дата обращения: 29.04.2010).
4. *A. Šilić J. C., Bašić B., Morin A.* N-grams and Morphological Normalization in Text Classification: a Comparison on a Croatian-English Parallel Corpus. [Электронный ресурс]. — SpringerLink. — URL: http://www.springerlink.com/content/t45u6740713896x5/ ; (дата обращения: 29.04.2010).
5. PeID [Электронный ресурс]. — URL: http://www.peid.info/ ; (дата обращения: 29.04.2010).



**Эдель Дмитрий Александрович**
Научный сотрудник ФГНУ НИИ «Спецвузавтоматика», г. Ростов-на-Дону.
Тел.: 8 (863) 201-28-22
Эл. адрес: sva@rsu.ru



D.A. Edel

**Language model of binary codes**


Current paper proposes comparison of the words of natural and machine languages based on the language model of executable code. This language model is constructed on basis of homogenous Markov chains. The results of experimental studies of proposed model are also provided.


**Keywords:** Machine code, Markov chain, gap completion.