НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

ISSN 1727-4907

П Р О Б Л Е М И ПРОГРАМУВАННЯ

НАУКОВИЙ ЖУРНАЛ

P R O B L E M S IN PROGRAMMING

SCIENTIFIC JOURNAL

2020

№ 1

Теми випуску:

- ***Теоретичні та методологічні основи програмування***
- ***Інструментальні засоби і середовища програмування***
- ***Моделі та засоби систем баз даних і знань***
- ***Методи та засоби комп´ютерного моделювання***
- ***Програмні системи захисту інформації***
- ***Мови програмування***

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

П Р О Б Л Е М И ПРОГРАМУВАННЯ

науковий журнал

Головний редактор
Андон Пилип Іларіонович
академік НАН України,
директор Інституту програмних систем
НАН України
✉ Інститут програмних систем
НАН України
проспект Академіка Глушкова, 40, корп. 5
03187, Київ-187
☎ Тел.+380 (44) 526 5507
E-mail: andon@isofts.kiev.ua
http://www.pp.isofts.kiev.ua

Адреса для кореспонденції

✉ Інститут програмних систем
НАН України
Проспект Академіка Глушкова, 40
03187, Київ-187

☎ Тел.: +380 (44) 526 5065
Факс: +380 (44) 526 6263
E-mail: iss@isofts.kiev.ua

Затверджено до друку вченою радою Інституту програмних систем НАН України.
Протокол № 3 від 06.02.2020 р.

Редактор *В.П. Замула*
Комп'ютерна верстка *В.П. Замула*

Підписано до друку 04.03.2020. Формат 60x84/8. Папір офс. Ум. друк. арк. 15,11.
Обл.-вид. арк. 12,41. Тираж 120 прим. Ціна договірна. Замовл.

Віддруковано ВД «Академперіодика» НАН України
вул. Терещенківська, 4, м. Київ, 01004

Свідоцтво суб'єкта видавничої справи ДК № 544 від 27.07.2001

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

ПРОБЛЕМИ ПРОГРАМУВАННЯ

науковий журнал

№ 1 січень – березень 2020

Заснований у березні 1999 р.

ЗМІСТ

Свідоцтво про державну реєстрацію КВ № 7490 від 01.07.2003

Науковий журнал “Проблеми програмування” занесений до переліку наукових фахових видань України, в яких можуть публікуватися основні результати дисертаційних робіт.

ISSN 1727-4907

NATIONAL ACADEMY OF SCIENCES OF UKRAINE
INSTITUTE OF SOFTWARE SYSTEMS

PROBLEMS IN PROGRAMMING

scientific journal

№ 1 January – March 2020

Founded in March, 1999

CONTENTS

УДК 004.8 https://doi.org/10.15407/pp2020.01.003

А.Ф. Кургаев

ОПИСАНИЕ СПИСКОВ И МНОЖЕСТВ В МЕТАЯЗЫКЕ НОРМАЛЬНЫХ ФОРМ ЗНАНИЙ

Предложена формализация списков, предикатов на списках и множествах в метаязыке нормальных форм знаний, базируясь на известных Пролог-формализациях этих понятий, использующих списковый домен. Среди предикатов на списках описаны: добавление элемента, удаление элемента, поиск последнего элемента, поиск соседних элементов, конкатенация списков, реверс и др. Используя списковый домен описаны предикаты на множествах: превращения списка в множество, принадлежности элемента множеству, объединения, пересечения, разности, симметрической разности, совпадения, дополнения множеств.

Ключевые слова: метаязык, список, множество, предикат, рекурсия, определение.

Введение

Список – один из самых простых и полезных типов структур составных объектов логического программирования, позволяющий во многих случаях улучшить «читабельность» программ. Запись в виде списка свободна по форме и одновременно достаточно точна и понятна [1–3].

Списки можно использовать для представления всевозможных знаний. Также они позволяют представить практически любые неоднородные и/или иерархические структуры данных, возможные в символьных вычислениях, поддерживающие функциональный стиль программирования. В виде списков удобно представлять формулы, функции, деревья, графы, множества и многие другие сложные объекты [4–6].

Множество – одна из наиболее важных структур данных, используемых как в математике, так и в программировании. Это набор элементов, подобный списку, отличающийся от него лишь фактом принадлежности элемента множеству [3].

В метаязыке нормальных форм знаний (НФЗ) [7, 8], в отличие от ЛИСПа и Пролога, нет такой встроенной структуры данных, как *список*, эффективность которой обоснована не только теоретически, но и обширной практикой использования этих языков при создании систем искусственного интеллекта. Тем более в метаязыке НФЗ нет такой встроенной структуры данных, как *множество*. Поэтому, для практического использования метаязыка НФЗ важно реализовать понятия списка, множества и предикаты на списках и множествах, опираясь на стандартные домены метаязыка НФЗ.

В качестве базовой структуры используем область данных из [7, 8]:

- домен *D* представлен двумя независимыми массивами – входным INP и выходным OUT, с которыми связаны переменные *m* и *n*, принимающие значения текущих координат соответствующих массивов;
- две библиотеки одноименных предикатов – библиотеку анализа над данными из INP и библиотеку порождения над данными из OUT;
- набор системных процедур, управляющих этими элементами домена *D*:
  - RB – истинный предикат переключения библиотек;
  - RIO – истинный предикат переключения массивов INP и OUT;
  - UIO – истинный предикат объединения / разделения массивов INP и OUT.

Во всех последующих формальных описаниях предикатов на списках и множествах использована нотация метаязыка НФЗ [7, 8]:

1. description = determination (determination) ;

2. determination = negativ nameConcept definition bodyDeterm endDeterm;
3. negativ = inversion / true;
4. nameConcept = identifier / integer / chainSigns;
5. identifier = letter (letter/decimalDigit);
6. integer = decimalDigit (decimalDigit);
7. chainSigns = ^metaSign sign (^metaSign sign);
8. bodyDeterm = structure / terminal;
9. terminal = space (space);
10. structure = singleDefinit (separator singleDefinit);
11. singleDefinit = negativ primary mode (concatenate negativ primary mode);
12. primary = iterationSeq / nameConcept / line;
13. iterationSeq = startIterationSymb bodyDeterm endIterationSymb;
14. mode = analysis / traceAnalysis / generation / true;
15. line = quotationMark nameConcept quotationMark;

где definition – разделитель двух частей определения, изображается символом '=';
separator – отношение альтернативного выбора изображается символом '/';
concatenate – отношение конкатенации изображается символом space ' ';
startIterationSymb, endIterationSymb – пара скобок '(' и ')', обрамляющих итерируемый элемент;
inversion – отношение отрицания изображается символом '^';
endDeterm – конец определения изображается символом ';';
quotationMark – текстовая кавычка, изображается символом '";
analysis – режим анализа изображается символом ‘?’;
traceAnalysis – режим анализа со следом изображается символом '#';
generation – режим порождения изображается символом '!';
letter = 'A' / 'B' / 'C' /…/ 'Z' / 'a' / 'b' / 'c' /... / 'z';
decimalDigit = '0'/'1'/'2'/'3'/'4'/'5'/'6'/'7'/'8'/'9';
sign = '-' / '&' / '%' / '$' / '@' / '~' / ':' / '<' / '>' / … / ',' / '.' / '_';
metaSign = '(' / ')' / space / '/' / '=' / '?' / '#' / '!' / ';' / '"/ ’”' ;

1. Представление задачи

В терминах метаязыка НФЗ любая задача формулируется как доказательство категорического суждения $P(x)$, предикат P которого задан именованной структурой, а субъект (возможно, многоместный аргумент x) задан последовательностью элементов, ограниченной круглыми скобками:

subject = "( )" / "(" element (","element) ")";

Структуру термина element примем подобной в Прологе и в нотации метаязыка НФЗ [7, 8] опишем так:

element = term / list;
list = '[' list_content ']';
list_content = element (',' element) / head comma tail / variable / true;
head = term (',' term);
comma = ',' / true;
tail = list;
term = number / variable / atom / structure;
structure = atom '(' term (',' term) ')';
variable = letter1 (letter / letter1 / number);
letter1 = A / B / C / ... / Z;
letter = a / b / c / ... / z;
number = numeral (numeral);
numeral = 0/1/2/3/4/5/6/7/8/9;
atom = letter (letter / letter1 / numeral);
E1= element;
E2= element;
E3= element;

Список – это рекурсивная структура данных, поэтому нужны и рекурсивные алгоритмы для его обработки. Главный способ обработки списка – это поэлементный просмотр и обработка списка до его исчерпания. Эти алгоритмы обычно задаются двумя утверждениями: одно определяет, что делать с пустым списком, второе – что делать с обычным списком.

При описании любой задачи необходимо, прежде всего, определиться с аргументами предиката, принять некоторую структуру области данных, описать про-

цесс анализа конкретного значения аргументов, и далее, – процесс вывода. Среди базовых предикатов на списках: формирование, объединение списков; поиск элемента в списке; вставка элемента в список и удаление из списка и др. [9].

2. Предикаты на списках

2.1. Печать списков. Печать элементов списка в Прологе задается двумя утверждениями:

```
write_a_list([ ]).
write_a_list([H|T]):- write(H), nl!, write_a_list(T).
/* nl - переход на новую строку */
```

Первое из них утверждает факт истинности для пустого списка, второе – инициирует печать головы непустого списка и рекурсивный вызов печати хвоста списка. В метаязыке НФЗ этот предикат определяется так:

```
write_a_list='(' '[' L# ']' ')' viv_write_a_list;
viv_write_a_list = ^genL ^X / ^stepL? ^write_H nl! viv_write_a_list;
^genL  = RIO L ']'! RIO;
^stepL =  X comma L# ;
^write_H = X nl!;
X = term;
L = term (',' term) / true;
```

Здесь, в первом утверждении описан анализ структуры аргумента предиката write_a_list, в процессе которого с переменной L связывается исходное значение списка, и далее, вызывается предикат viv_write_a_list, первая альтернатива которого утверждает, что пустой список печатать не надо, а вторая альтернатива утверждает поэлементную печать головы списка и рекурсивный вызов предиката viv_write_a_list на сокращенном списке.

2.2. Добавление элемента в список. У предиката add(X,L,L1]) три аргумента: добавляемый элемент X, начальный список L и результирующий – L1. Проще всего добавить элемент в список – вставить его в самое начало, как голову нового списка L1. В Прологе это записывают в виде факта:

```
add(X,L,[X|L]).
```

Все варианты вывода предиката add(X,L,L1) (добавление: известного элемента X к пустому или непустому списку L с получением неизвестного списка L1; неизвестного элемента X к известному списку L с получением известного списка L1; неизвестного элемента X к неизвестному списку L с получением известного списка L1; известного элемента X к известному списку L с получением известного списка L1) в метаязыке определим в форме пяти альтернативных определений термина add:

```
add = '(' X# ',' '[' ']' ',? '[' A# ']' ')' ^writeXS /
      '(' X# ',' '[' T# ']' ',? '[' A# ']' ')' ^writeAdd /
      '(' A# ',' '[' L1# ']' ',? '[' L2# ']' ')' ^genL2
       controlAdd ^writeX /
       '(' A# ',' '[' B# ']' ',? '[' L2# ']' ')' ^genL2
        bindingX_L1 ^writeVV /
       '(' X# ',' '[' L1# ']' ',' '[? L2# ']' ')' ^genL2
       analysX_L1;
^writeXS = A '=' '[' X ']' nl!;
^writeAdd = A '=' '[' X ',' T ']' ';' nl!;
^genL2  = RIO L2 ']'! RIO;
controlAdd = X# analysL1 ^',';
analysL1 = RB L1! RB / ^RB;
bindingX_L1 = X comma L1# ^',';
^writeX = A '=' X ';' nl!;
^writeVV = A '=' X ',' B '=' L1 ';' nl!;
analysX_L1 = RB X comma L1! RB / ^RB;
A = variable;
B = variable;
T = term (',' term);
L1 = term (',' term) / true;
L2 = term (',' term) / true;
```

Первая альтернатива завершается порождением одноэлементного списка, вторая альтернатива – порождением нового списка, составленного из известного элемента с присоединением к нему известного списка. Третья альтернатива после успешного вычитания первого списка из второго завершается порождением оставшейся головы второго списка. Четвертая альтернатива после успешного разделения второго списка на его голову и хвост завершается порождением найденных элемента и первого списка. Пятая альтернатива состоит в проверке на эквивалентность

второго списка с конкатенацией известного элемента и первого списка.

2.3. Удаление элемента. Предикат away(X,L,L1) удаления элемента оперирует тремя аргументами: L1 – это список L, из которого изъят элемент X. В Прологе записывается двумя утверждениями, первое из которых – простой факт, завершающий вычисление, а второй – рекурсивное правило:

```
away(X, [X|T],T).
away(X, [Y|T], [Y|T1]):- away(X,T,T1).
```

Все четыре варианта (удаление известного элемента X из известного списка L с получением неизвестного списка L1; выяснение, действительно ли известный список L1 получен удалением известного элемента X из известного списка L; выяснение неизвестного списка L, из которого удален известный элемент X с получением известного списка L1; выяснение неизвестного элемента X, удаленного из известного списка L с получением известного списка L1) вывода предиката away(X,L,L1) в метаязыке определим в форме соответствующих четырех альтернатив.

```
away = away1 /  away2 /  away3 / away4;
away1='(' X# ',' '['? T# ']' ','? A L# ')'
viv_away1 ^writeL;
viv_away1 = ^genT analysX ','? T#
^form_result / ^current_result viv_away1;
away2='(' X# ',' '['? T# ']' ',' '['? L1 L# ']' ')'
viv_away1 ^genL analysL1;
away3='(' X# ','? A# ',' '['? L1# ']' ')' ^genXL1?
L# ^writeL;
away4='(' A# ',' '['? T# ']' ',' '['? L# ']' ')'
viv_away2;
viv_away2 = ^genT ^headTail1 ^genL
searchMatches viv_away2 / ^writeX;
searchMatches = analysX / Y comma
searchMatches;
 ^headTail1  = X ',' T#;
^genXL1 = RIO X ',' L1! RIO;
^form_result = ^genL variantsL? L#;
variantsL = ^T? ^genT? / ^concatL_T;
^concatL_T = RIO L ',' T RIO!;
^current_result = Y ',' T# ^genL currentL? L#;
currentL = ^T? ^genY / ^concatL_Y;
^concatL_Y = RIO L ',' Y RIO!;
^genT = RIO T ']'! RIO;
^genY  = RIO Y ']'! RIO;
^writeL = A '=' '[' L ']' nl!;
analysX = RB X! RB / ^RB;
Y = term;
```

2.4. Принадлежность элемента списка. У предиката member(X,L) принадлежности элемента X списку L два аргумента: L – некоторый список и X – объект того же типа, что и элементы списка L. Его определение основывается на следующем: X – или голова списка L, или X принадлежит его хвосту. В Прологе это записывают в виде двух утверждений, первое из них – факт, завершающий вычисление, второй – рекурсивное правило. Если же список пуст, то предикат ложен: у пустого списка нет элементов:

```
member(X, [X| _ ]).
member(X, [ _ | T ]):- member(X,T).
```

В метаязыке предикат member(X,L) можно определить рекурсией: если X совпадает с головой списка, то, независимо от значения его хвоста, получим результирующее значение истинности «истина»; иначе, делается очередной шаг исследования следующего элемента списка L после удаления головы текущего списка.

```
member = '(' X# ',' '[' T# ']' ')' viv_member /
   '(' A# ',' '['? X# ^writeX (',' X# ^writeX) ']'
')';
viv_member = ^genT  analysX /
     ^step2 viv_member;
^step2 = Y comma? T#.
```

Этот предикат можно использовать так: во-первых, конечно, для проверки, есть ли в списке конкретное значение, во-вторых, – получить элементы заданного списка.

2.5. Конкатенация списков. У предиката conc(L1,L2,L3) конкатенации списков три аргумента; первые два L1 и L2 – объединяемые списки, а третий – список L3 результат конкатенации первых двух. За основу определения этого термина берут рекурсию по первому списку, а за базис – факт, утверждающий, что присоеди-

нение произвольного списка к пустому списку дает тот самый произвольный список.

```
conc([ ], L, L).
conc([H|T], L, [H|T1]) :- conc(T,L,T1).
```

Этот предикат можно применять для решения разных задач: для получения списка, являющегося конкатенацией двух заданных; для проверки, является ли третий список конкатенацией двух первых; для разбивки третьего списка на подсписки; для поиска одного из первых списков по заданным двум другим и др. Для этих вариантов использования предикат конкатенации conc(L1,L2,L3) определим в метаязыке так:

```
conc=conc_1/conc_2/conc_3/conc_4/conc_5;
conc_1 = '(' '[' ']' ',' '[' ']' ',' analysA1 ')' /
      '(' '[' L1# ']' ',' '[' ']' ',' analysA2 ')' /
      '(' '[' ']' ',' '[? L2# ']' ',' analysA3 ')' /
      '(' '[? L1# ']' ',' '[? L2# ']' ',' analysA4 ')';
analysA1 = A# ^writeEmpty / '[' ']';
^writeEmpty = A '=' '[' ']' nl!;
analysA2 = A# ^writeL1 / analysL1 ^',';
^writeL1 = A '=' '[' L1 ']' nl!;
analysA3 = A# ^writeL2 / analysL2 ^',';
^writeL2 = A '=' '[' L2 ']' nl!;
analysL2 = RB L2! RB / ^RB;
analysA4=A# ^writeL1_L2 / analysL1_L2 ^',';
^writeL1_L2 = A '=' '[' L1 ',' L2 ']' nl!;
analysL1_L2 = RB L1 ',' L2! RB / ^RB;
conc_2 = '(? A# ',' '[? L2# ']' ',' '[? L3# ']' ')'
          ^genL3 analysCon6;
^genL3 = RIO L3 ']'! RIO;
analysCon6 = analysL2 ^',' ^writeEmpty /
  ^headTail3 ^genX ^takeL1 ^genL3 analysC;
^headTail3 = X ',' L3#;
analysC = analysL1_L2 ^',' ^writeL1 /
^headTail3 ^genL1X ^takeL1 ^genL3 analysC;
^genL1X = RIO L1 ',' X ']'! RIO;
conc_3 = '(' '[? L1# ']' ',? A# ',' '[? L3# ']' ')'
          ^genL3 analysL1 writeConc5;
writeConc5=^',' ^writeEmpty / ',' L2# ^writeL2;
conc_4 = '(? A L1# ',? B# ',' '[' L3# ']' ')'
^genL3 ^takeL2 ^writePair ^headTail2 below;
below = ^genX ^takeL1 ^writePair (^genL2 ^headTail2 ^genL1X ^takeL1 ^writePair);
^takeL2= L2#;
^headTail2 = X comma L2#;
^takeL1 = L1#;
^writePair = A '=' '[' L1 ']' ',' B '=' '[' L2 ']' ';' nl!;
conc_5 = '(? A L1# ',' '[? Y ',' B# ']' ',' '['
          zeroPair nextPair;
zeroPair = T3# ']' ')' ^genT3 ^listL2?
           ^writePair / ^takeXL2 ^genX;
nextPair = ^takeL1 ^genL2 ^listL2 ^writePair /
           ^takeXL2 ^genL1X nextPair;
^listL2= analysY comma? L2#;
analysY = RB Y! RB / ^RB;
^takeXL2 = X comma L2#;
^genT3 = RIO T3 ']'! RIO;
^genX = RIO X ']'! RIO;
L3 = term (',' term) / true;
L4 = term (',' term) / true;
T1 = term (',' term);
T2 = term (',' term);
T3 = term (',' term);
T4 = term (',' term).
```

Первая альтернатива conc_1 определяет варианты формирования результата, если первые два списка составлены из констант или один из них или оба являются пустыми, а третий список задан константами или именован переменной. Вторая альтернатива conc_2 определяет вариант поиска первого списка, если второй и третий списки составлены из констант. Третья альтернатива conc_3 определяет вариант поиска второго списка, если первый и третий списки составлены из констант. Четвертая альтернатива conc_4 определяет вариант поиска первого и второго списков, если третий список составлен из констант. Пятая альтернатива conc_5 определяет вариант поиска первого A и второго B списков, находящихся соответственно левее и правее заданного элемента Y третьего списка '[' T3 ']', составленного из констант.

2.6. Последний элемент списка. В Прологе предикат last(L,X) определяется как последний элемент одноэлементного списка являясь этим элементом, а послед-

ний элемент обычного списка – это последний элемент его хвоста:

```
last ([X],X).
last ([_|L],X):- last (L,X).
```

В метаязыке этот предикат можно определить, используя итерацию:

```
last = '(' '[' (Y ',' )? X# ']' ','? A# ')' ^writeX.
```

Здесь итерация (Y ',') выбирает первые *n*-1 элементов списка L , последний элемент которого связывает переменную X#, чье значение далее составляет результат решения.

2.7. Два соседних элемента списка. У этого предиката neighbors(X,Y,L) три параметра: первые два X, Y – элементы, третий L – список элементов такого же типа. Для случая, когда порядок размещения элементов X,Y в списке L важен, в Прологе этот предикат определяют с использованием предиката конкатенации:

```
neighbors(X,Y,L):- conc(_,[X,Y|_],L).
```

Все пять вариантов (выяснение, действительно ли известные два элемента X, Y соседствуют в известном списке L; выяснение, какой неизвестный элемент X является соседним слева от известного элемента Y в известном списке L; выяснение, какой элемент Y является соседним справа от известного элемента X в известном списке L; выяснение, какие пары элементов X, Y соседствуют в известном списке L; сконструировать новый список из двух известных элементов) вывода предиката neighbors(X,Y,L) в метаязыке определим в форме соответствующих альтернатив.

В метаязыке предикат neighbors(X,Y,L) можно определить так:

```
neighbors = neighbors1 /  neighbors2 /
        neighbors3 / neighbors4 / neighbors5;
neighbors1 = '(' X# ',' Y# ',' '[' L# ']' ')'
            viv_neighbors1;
viv_neighbors1 = ^genL analysXY / ^stepL
            viv_neighbors1;
neighbors2='(' X# ','? Y# ','? A# ')'
^writeXYS;
neighbors3 = '(' X# ','? A# ',' '['? L# ']'  ')'
            viv_neighbors3;
viv_neighbors3 = ^genL  analysX? Y#
^writeY / ^stepL viv_neighbors3; ;
neighbors4='(' A# ','? Y# ',' '['? L# ']' ')'
            viv_neighbors4;
viv_neighbors4 = ^genL?  X# analysY
^writeX / ^stepL viv_neighbors4;
neighbors5='(' A# ','? B# ',' '['? L# ']' ')'
(^genL?
    X comma Y# ^writeXY ^genL ^stepL);
analysXY = RB X ',' Y! RB / ^RB;
^writeXYS = A '=' '[' X ',' Y ']' nl!;
^writeY = A '=' Y ';' nl!;
^writeXY = A '=' X ',' B '=' Y ';' nl!.
```

2.8. Реверс списка. У этого предиката reverse(L1, L2) два параметра: первый – исходный список L1, второй – обращенный список L2. За базис определения этого термина принимают факт, утверждающий, что реверс пустого списка дает пустой список. Шаг рекурсии состоит в конкатенации обращенного хвоста списка с первым элементом исходного списка:

```
reverse([ ],[ ]).
reverse([X|T],Z):–  reverse(T,S),
conc(S,[X],Z).
```

В метаязыке предикат reverse(L1, L2) можно определить так:

```
reverse = '(' '[' ']' ',' A# ')' ^writeEmpty /
    '(' '[' L2# ']' ',' A# ')' viv_reverse ^writeL1 /
     '(' '[' L2# ']' ',' '['? L3# ']' ')' v_reverse
     ^genL1  analysL3;
viv_reverse = ^genL2 ^headTail2  ^genX
     ^takeL1 (^genL2 ^headTail2 ^genL1X
     ^takeL1);
analysL3 = RB L3! RB / ^RB.
```

2.9. Палиндром. Палиндромом называют список, совпадающий со своим обращением. В Прологе предикат palindrom(L) определяют через предикат reverse (L,L) с двумя одинаковыми аргументами:

```
palindrom(L):- reverse (L,L).
```

Подобное метаязыковое определение состоит из двух последовательных преобразований: обращение заданного

списка и проверка, совпадает ли результат с начальным списком:

```
palindrom = '(' '[' L2# ']' ')' viv_reverse ^genL1 analysL2.
```

2.10. Удаление из списка всех вхождений заданного значения. Этот предикат delete_all(X,[L],[L1]) имеет три параметра: первый X – удаляемый элемент, второй L – начальный список, а третий L1 – результат удаления из начального списка L всех вхождений заданного элемента X.

Определение на Прологе включает три утверждения: первое является базовым фактом – пустой список не уменьшается; второе утверждает, что после удаления заданного элемента из головы начального списка L результирующий список является хвостом списка L без вхождений заданного элемента X; третье утверждает, что результирующий список является конкатенацией последовательности элементов без удаляемого и результата удаления заданного элемента из хвоста начального списка.

```
delete_all(_,[],[]).
delete_all(X,[X|L],L1):- delete_all (X,L,L1).
delete_all (X,[Y|L],[Y|L1]):- X<>Y, delete_all (X,L,L1).
```

Метаязыковое определение для неизвестного результирующего списка (заданного именем A переменной) включает две рекурсивные альтернативы: удаление головы текущего списка L2 в случае ее тождественности заданному элементу; иначе присоединить голову текущего списка к текущему составу результирующего списка L3.

```
delete_all = '(' Y# ',' '[' L2# ']' ',' '[' A L3# ']' ')'
             viv_del_all;
viv_del_all = ^genL2 ^listL2 viv_del_all / ^takeXL2 ^upbuildingL3 ^takeL3 viv_del_all / ^writeL3;
^genL3X  = RIO L3 ',' X ']'! RIO;
^upbuildingL3 = ^genL3 ^nullX ^genL3X / ^genX;
^nullX = X;
^takeL3= L3#;
^writeL3 = A '=' '[' L3 ']' nl!.
```

Если нужно изъять не все вхождения определенного значения в список, а только первое, то в Прологе используют следующее определение:

```
delete_one(_,[],[]).
delete_one(X,[X|L],L):-!.
delete_one(X,[Y|L],[Y|L1]):-
      delete_one(X,L,L1).
```

Метаязыковое определение этой семантики получает вид:

```
delete_one = '(' Y# ',' '[' L2# ']' ',' '[' A L3# ']' ')' viv_del_one;
viv_del_one = ^genL2 ^listL2 ^writeL3_L2 /
              ^takeXL2 ^upbuildingL3 ^takeL3
              viv_del_one/^writeL3;
^writeL3_L2 = A '=' '[' L3 ',' L2 ']' nl!.
```

3. Понятие множества

В метаязыке НФЗ, также как в Прологе и ЛИСПе нет такой встроенной структуры данных, как *множество*. Поэтому, придется реализовывать это понятие, используя имеющиеся стандартные домены. В качестве базового примем списковый домен, приняв:

Определение 1. Множество – список без элементов повтора.

Фактически, нет никакой реализации понятия множество, которое бы точно отвечало этому математическому объекту. Принятое определение – лишь приближение к "истинному" объекту множество.

Рассмотрим предикаты, реализующие основные теоретико-множественные отношения.

3.1. Предикат превращения списка в множество. Эту функцию выполняет предикат, удаляющий из списка все повторные вхождения элементов. Аргументами предиката unik(L1,L2) являются два списка: произвольный начальный – L1 и результирующий – L2 без повторяемых элементов. Его семантика простая: если элемент X головы списка L1 входит в хвост этого списка L1 (проверяется предикатом member), то он не записывается в результирующий список L2, иначе этот элемент X списка L1 вставляется в результирующий список L2 в качестве его головы.

Пролог описание этого предиката имеет вид.

```
unik([ ],[ ]).
unik([H|T], L):– member(H,T), unik(T,L).
unik([H|T], [H|L]):– unik(T,L).
```

Подобное метаязыковое определение включает две альтернативы: первая фиксирует тот факт, что пустой список остается пустым безусловно; вторая альтернатива определяет процесс вывода результирующего списка для общего случая в форме итерации двух вложенных альтернатив. Первая из них определяет наличие повторений в текущем исследовании списка, а вторая – процесс пошагового накопления результирующего списка с использованием термина ^upbuildingL1.

```
unik = '(' '[' ']' ',' '[' ']' ')' / '(' '[' L# ']' ',? A L1#
   ')' viv_unik ^writeL1;
viv_unik=(^genL ^headTail2 member1
^genL ^stepL/ ^genL ^stepL ^upbuildingL1
^takeL1);
member1 = ^genL2 analysX / ^stepL2
          member1;
^upbuildingL1 = ^genL1 ^nullX ^genXL1 /
                ^genX;
^genL1 = RIO L1 ']'! RIO;
^stepL2 = Y ',? L2# ']';
```

Далее рассмотрим метаязыковую реализацию основных теоретико-множественных отношений.

3.2. Принадлежность элемента множеству. В качестве реализации предиката принадлежности элемента *x* множеству A ($x \in A$) можно использовать предикат member(X,L) принадлежности элемента X списку L, где: X – объект того же типа, что и элементы списка L.

3.3. Объединение двух множеств. Объединением двух множеств является множество, чьи элементы принадлежат или первому, или второму множеству. Формальное определение объединения множеств A и B:

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$

на рис. 1 обозначено штриховкой.

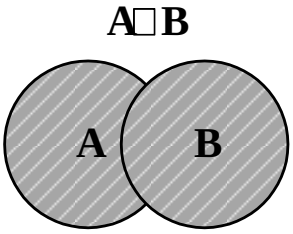


Рис. 1. Объединение множеств A и B

У предиката union(L1,L2,L3) три параметра: первые два L1, L2 – объединяемые множества, третий L3 – результат объединения двух первых множеств. При этом важно, чтобы ни одно значение не входило в итоговое множество повторно.

Пролог описание этого предиката имеет вид.

```
union([ ],S2,S2).
union([H|T],S2,S):– member(H,S2), !,
                    union(T,S2,S).
union([H|T],S2,[H|S]):– union(T,S2,S).
```

В метаязыке предикат union(L1,L2,L3) определим для случаев, когда являются пустыми оба первых множества или один из них, или же оба первых множества составлены из констант, а третье множество является, соответственно, пустым, задано константами или именовано переменной.

```
union = '(' '[' ']' ',' '[' ']' ',' analysA1 ')' /
      '(' '[' L1# ']' ',' '[' ']' ',? A# ')' ^writeL1 /
      '(' '[' ']' ',' '[? L1# ']' ',? A# ')' ^writeL1 /
      '(' '[? L# ']' ',' '[' ']' ',' '[? L1# ']' ')'
       equivalentLL1 / '(' '[' ']' ',' '[? L# ']' ',' '[?
       L1# ']' ')' equivalentLL1 / '(' '[? L# ']' ','
 '[? L1# ']' ',' A# ')' viv_union_2 ^writeL_L1 /
      '(' '[? L# ']' ',' '[? L1# ']' ',' '[? L4# ']' ')'
      viv_union_3 / '(' '[? L1# ']' ',? A# ',' '[?
      L2# ']' ')' viv_minus_L2L1 ^writeL /
       '(? A# ',' '[? L1# ']' ',' '[? L2# ']' ')'
 viv_minus_L2L1 ^writeL / '(? A L1# ',? B#
',' '[' L3# ']' ')' ^genL3 ^takeL2 ^unionAnswer;
^unionAnswer = ^writePair ^headTail2
        ^hypothes1 ^takeL1 ^writePair below;
below = (^genL2 ^headTail2 ^hypothes2
          ^takeL1 ^writePair);
equivalentLL1 = ^genL (^headTail2 ^genL1
      ^takeL3 member2 ^genL2) nullX
      ^genL1 (^headTail2 ^genL ^takeL3
```

```
        member2 ^genL2) nullX;
member2 = ^genL3 analysX / ^stepL3
        member2;
viv_ union_2 = ^genL (^headTail2 ^genL1
            ^takeL31 member3 ^genL2) nullX;
member3 = (^genL3  analysX ^takeL3 /
            ^stepL3 ^genYL1 ^takeL1);
viv_ union_3 = viv_ union_2 ^genLL1? L#
            ^genL4?  L1# equivalentLL1;
^writeL_L1 = A '=' '[' L ',' L1 ']' nl!;
^takeL31 = L3 L1#;
^stepL3 = Y ',' L3#;
^genYL1 = RIO Y ',' L1! RIO;
^genLL1  = RIO L ',' L1 ']'! RIO;
^genL4  = RIO L4 ']'! RIO.
```

Интерпретация термина analysA1 вывода результата для пустых объединяемых множеств состоит в порождении пустого множества, как значения переменной A, или порождении значения истинности истина, если результирующее множество задано формой пустого множества.

Вторая и третья альтернативы завершаются печатью writeL1 значения известного множества в качестве значения результирующего множества.

Четвертая и пятая альтернативы завершаются значением истинности истина термина viv_union_1, если эквивалентны результирующее и не пустое заданное множество.

Шестая альтернатива завершается печатью writeL_L1 значения объединенного множества после удаления из множества L1 элементов множества L в процессе интерпретации термина viv_union_2.

Седьмая альтернатива в процессе вывода термина viv_union_3 состоит вначале в синтезе объединения множеств L и L1 (используя предикат viv_union_2), далее переназначение переменных и, в завершение (используя предикат equivalentLL1), вывод эквивалентности синтезированного множества заданному значению результирующего множества.

Восьмая и девятая альтернативы вычисляют и печатают неизвестное множество, используя термин viv_minus_L2L1 вычитания над известными (результирующим и одним из компонентных) множествами.

Десятая альтернатива состоит в порождении перераспределения заданного значения результирующего множества на два составляющих множества. Семантика этого процесса совпадает с семантикой термина conc_4 порождения перераспределения заданного значения результирующего списка на два подсписка.

3.4. Пересечение двух множеств. Пересечение двух множеств – это множество из элементов, принадлежащих и первому, и второму множествам. Формальное определение пересечения множеств A и B:

$$A \cap B = \{x | x \in A \ \& \ x \in B\}$$

на рис. 2 обозначено штриховкой.

У предиката, реализующего эту операцию, три параметра: первые два L1, L2 – исходные множества, третье множество L3 – результат пересечения двух первых.

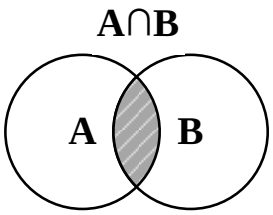


Рис. 2. Пересечение множеств A и B

В итоговое множество входят те элементы, которые есть и в первом, и во втором множестве одновременно. Пролог описание этого предиката имеет вид.

```
intersection([],_,[]).
intersection([H|T1],S2,[H|T]):–
        member(H,S2),!,
intersection(T1,S2,T).
intersection([_|T],S2,S):–
intersection(T,S2,S).
```

Базис рекурсии: пересечение пустого множества с любым множеством будет пустым множеством.

Шаг рекурсии включает два случая: если голова первого множества является элементом второго множества, она становится головой результирующего множества, чей хвост образуется как пересечение хвоста первого множества со вторым множеством; иначе, результирующее множе-

ство образуется пересечением хвоста первого множества со вторым множеством.

В метаязыке предикат intersection(L1,L2,L3) определим для случаев, когда пустыми являются оба первых множества или одно из них, или же оба первых множества составлены из констант, а третье множество является, соответственно, пустым, задано константами или именовано переменной.

```
intersection = intersectionEmpty ',' analysA1
       ')' / '(' ['? L1# ']' ',' ['? L2# ']' ',? A L# ')'
       viv_intersect_1 ^writeL / '(' ['? L1# ']' ','
    ['? L2# ']' ',' ['? L4# ']' ')' viv_intersect_2;
intersectionEmpty = '(' ['  ']' ',' ['  ']' /
            '(' ['  L1 ']' ',' ['  ']' / '(' ['  ']' ',' ['  L1 ']';
viv_intersect_1 = (^genL2 ^headTail2 ^genL1
          ^takeL3 member4) nullX;
member4 = ^genL3  analysX ^genXL? L# /
           ^stepL3 member4;
viv_intersect_2 = viv_intersect_1 ^genL4?
             L1# equivalentLL1;
^genXL = RIO X ',' L! RIO.
```

Интерпретация термина analysA1 вывода результата при наличии пустых множеств состоит в порождении пустого множества, как значения переменной A, или порождении значения истинности истина, если результирующее множество задано формой пустого множества.

Вторая альтернатива завершается печатью writeL множества, являющегося пересечением заданных множеств.

Третья альтернатива описывает последовательность действий: вначале, синтез пересечения двух заданных множеств, далее переназначение переменных и, в завершение, вывод эквивалентности синтезированного множества заданному значению результирующего множества.

3.5. Разность двух множеств. Разность двух множеств – это множество, составленное из элементов первого множества, не принадлежащих второму. На рис. 3 штриховкой обозначена формальная запись двух вариантов разности множеств A и B:

$$A\backslash B=\{x|x\in A \ \& \ x\notin B\},$$

$$B\backslash A=\{x|x\in B \ \& \ x\notin A\}.$$

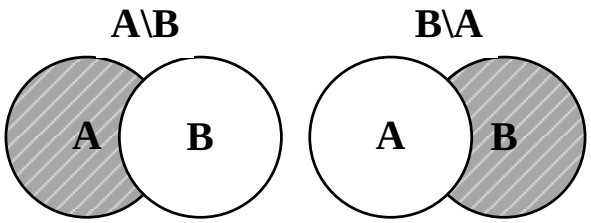


Рис. 3. Разность множеств A и B

У предиката minus(L1,L2,L3), реализующего разность, три аргумента: первый L1 – уменьшаемое множество, второй L2 – вычитаемое множество, третий L3 – результат вычитания второго множества из первого. Третье множество содержит элементы первого множества без элементов второго. Пролог-описание этого предиката имеет вид.

```
minus([],_,[]).
minus([H|T],S2,S):– member(H,S2),!,
          minus(T,S2,S).
minus([H|T],S2,[H|S]):– minus(T,S2,S).
```

Базис рекурсии: вычитание произвольного множества из пустого множества ничего, кроме пустого множества, дать не может. Шаг рекурсии: если голова первого множества входит во второе множество, то разность множеств образуется вычитанием второго множества из хвоста первого; иначе, разность множеств является конкатенацией головы первого множества и результата вычитания второго множества из хвоста первого множества.

В метаязыке предикат minus(L1,L2,L3) определим для случаев, когда: пустыми являются оба первых множества или только уменьшаемое; пустым является вычитаемое множество, а результирующее неизвестно; пустым является вычитаемое множество, а уменьшаемое и результирующее множества заданы; заданы уменьшаемое и вычитаемое множества, а результирующее неизвестно; все три множества заданы или же заданы вычитаемое и результирующее множества, а уменьшаемое неизвестно.

```
minus ='(' ['  ']' ',' ['? L1# ']' ',' analysA1 ')' /
       '(' ['  L1# ']' ',' ['  ']' ',? A# ')' ^writeL1 /
       '(' ['  L# ']' ',' ['  ']' ',? L1# ')'
```

```
        equivalentLL1 /
        '(' '[? L2# ']' ',' '[? L1# ']' ',' A L # ')'
        viv_minus_L2L1 ^writeL /
        '(' '[? L2# ']' ',' '[? L1# ']' ',' '[? L4# ']' ')'
        viv_minus /
        '(' '[? A# ']' ',' '[? L# ']' ',' '[? L1# ']' ')'
        ^writeL_L1 /
        '(' '[? L2# ']' ',' A L # ',' '[? L1# ']' ')'
        viv_minus_L2L1 ^writeL;
viv_minus_L2L1 = (^genL2 ^headTail2
        stepMinus);
stepMinus = ^genL1 ^takeL3 ^member5
        ^genXL? L# / true;
member5 = ^genL3 analysX/^stepL3
         member5;
viv_minus = viv_minus_L2L1 ^genL4? L1#
         equivalentLL1.
```

Интерпретация термина analysA1 вывода результата при наличии пустых множеств состоит в порождении пустого множества, как значения переменной A, или порождении значения истинности истина, если результирующее множество задано формой пустого множества.

Вторая альтернатива завершается печатью writeL уменьшаемого множества, как значения результирующего множества.

Третья альтернатива описывает условие эквивалентности уменьшаемого и результирующего множеств при пустом вычитаемом множестве.

Четвертая альтернатива описывает последовательность действий по формированию и печати разности заданных множеств: уменьшаемого и вычитаемого.

Пятая альтернатива описывает вычисление разности заданных множеств, далее переназначение переменных и, в завершение, вывод эквивалентности синтезированного множества заданному значению результирующего множества.

Шестая альтернатива завершается печатью writeL_L1 конкатенации вычитаемого и результирующего множеств.

Седьмая альтернатива описывает последовательность действий по формированию и печати разности заданных множеств: уменьшаемого и результирующего.

3.6. Отношение подмножества. Одно множество содержится в другом, если каждый элемент первого множества принадлежит второму множеству. Формальное определение отношения подмножества (AKO – a kind of) множеств A и B:

$$A \subseteq B \Leftrightarrow \forall x(x \in A \Rightarrow x \in B).$$

На рис. 4 обозначено штриховкой.

У предиката subset(L1,L2), реализующего это отношение, два параметра: L1 – множество, являющееся подмножеством другого L2 множества.

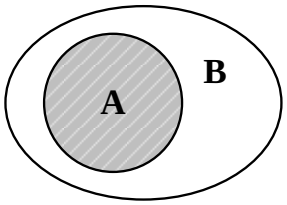


Рис. 4. Множество A является подмножеством B

Базис рекурсии: представлен фактом, утверждающим, что пустое множество является подмножеством любого множества.

Шаг рекурсии: первое множество является подмножеством второго, если первый элемент первого множества принадлежит второму множеству, а его хвост является подмножеством второго множества.

Пролог описание этого предиката имеет вид.

```
subset([],_).
subset([H|T],S):– member(H,S), subset(T,S).
```

В метаязыке предикат subset(L1,L2) определим для трех случаев, когда: пустым является первое L1 множество, а второе L2 – любым; оба множества L1, L2 – произвольны; первое L1 множество задано переменной, а второе – константное.

```
subset =  '(' '[' ']' ',' '[' L1 ']' ')' / '(' '[' L2# ']' ',' '['
        L1# ']' ')' viv_Subset / '(' A L1#  ','?
        '[' L# ']' ')' ^writeL1 ^genL ^headTail2
   ^hypothes1 ^takeL1 ^writeL1 viv_Subset1;
viv_Subset = (^genL2 ^headTail2 stepSubset)
        nullX;
stepSubset = ^genL1 ^takeL3 ^member5;
```

```
viv_Subset1 = (^genL2 ^headTail2
        ^hypothes2  ^takeL1 ^writeL1).
```

Первая альтернатива определения термина subset фиксирует тот факт, что пустое множество является подмножеством любого множества.

Вторая альтернатива состоит в поочередной проверке элементов первого множества L2 со всеми элементами второго множества L1.

Третья альтернатива описывает процесс последовательного порождения всех (начиная с пустого) подмножеств заданного множества L2.

3.7. Совпадение двух множеств. Два множества A и B называются равными, если одновременно выполнено $A \subseteq B$ и $B \subseteq A$, т. е. множество A содержится в множестве B и множество B в множестве A. Иначе говоря, два множества равны, если все элементы первого множества содержатся во втором множестве, и наоборот. Пролог-описание предиката, описывающего отношение эквиваленции двух множеств, имеет вид.

```
equal(A,B):– subset(A,B), subset(B,A).
```

В метаязыке предикат equal(L,L1) определяется последовательностью итераций: первая проверяет вхождение всех элементов первого множества L во второе множество L1, а вторая итерация проверяет вхождение всех элементов второго множества L1 в первое множество L.

```
equal = '(' '[? L# ']' ',' '[? L1# ']' ')' subsetLL1
        subsetL1L;
subsetLL1= ^genL (^headTail2 ^genL1
       ^takeL3 member2 ^genL2) nullX;
subsetL1L = ^genL1 (^headTail2 ^genL
       ^takeL3 member2 ^genL2) nullX.
```

3.8. Отношение собственного подмножества. Если множество B содержит множество A и другие элементы, кроме элементов множества A, то A – собственное подмножество множества B: $A \subset B$ (рис. 4) на Прологе имеет вид.

```
prop_subset(A,B):– subset(A,B),
                   not(equal(A,B)).
```

Подобным же образом этот предикат prop_subset(L,L1) определяется и в метаязыке:

```
prop_subset = '(' '[? L# ']' ',' '[? L1# ']' ')'
                   subsetLL1 ^subsetL1L.
```

3.9. Симметрическая разность. В отличие от обычной разности, симметрическая разность не зависит от порядка ее аргументов. Симметрической разностью двух множеств называется множество, чьи элементы либо принадлежат первому и не принадлежат второму множеству, либо принадлежат второму и не принадлежат первому множеству. Формальное определение симметрической разности множеств A и B:

$$A \Delta B = \{x | (x \in A \ \& \ x \notin B) \vee (x \in B \ \& \ x \notin A)\}.$$

На рис. 5 обозначено штриховкой.

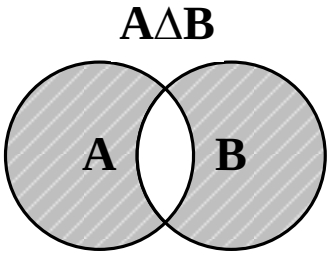


Рис. 5. Симметрическая разность множеств A и B

Симметрическую разность можно выразить через уже реализованные операции:

$$A \Delta B = (A \setminus B) \cup (B \setminus A).$$

Эти соотношения на Прологе получат вид.

```
sim_minus(A,B,SM):– minus(A,B,A_B),
        minus(B,A,B_A),
union(A_B,B_A,SM).
```

В метаязыке предикат sim_minus(L1,L2,L3) также можно выразить через уже определенные предикаты minus(L1,L2,L3) и union(L1,L2,L3) в следующей форме.

```
sim_minus = '(' E1 ',' E2 ',' E3# ')'
```

```
    ^genE1E2AB minus takeAB
    ^genE2E1BA minus takeBA
   ^genABBASM union;
^genE1E2AB = '(' E1 ',' E2 ',' 'AB' ')'!;
takeAB = 'AB' '=' '['? A_B#;
^genE2E1BA = '(' E2 ',' E1 ',' 'BA' ')'!;
takeBA = 'BA' '=' '['? B_A#;
^genABBASM = '(' A_B ',' B_A ',' 'SM' ')'!.
```

В этом выражении термины ^genE1E2AB, ^genE2E1BA и ^genABBASM формируют многоместные аргументы соответствующих предикатов minus и union, а takeAB и takeBA – принимают вычисленные значения переменных.

3.10. Дополнение. Дополнением DA множества A является подмножество элементов некоторого определенного универсума U, не принадлежащих A. Формальное определение отношения supp(A,DA) дополнения:

$$DA = U \backslash A = \{x | x \in U \ \& \ x \notin A\}.$$

На рис. 6 обозначено штриховкой.

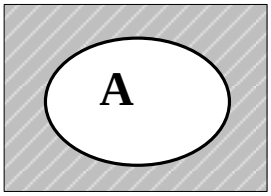


Рис. 6. Отношение дополнения множества A

Множество DA вычислимо лишь тогда, когда определены множества U и A. Для этого случая семантика предиката supp(U,A,DA) совпадает с семантикой предиката minus(U,A,DA), что на Прологе представляется в форме.

```
supp(U,A,DA):– minus(U,A,DA).
```

Предикат supp(L1,L2,L3) можно выразить и в метаязыке через уже определенный предикат minus(L1,L2,L3) в следующей форме.

```
supp = '(' E1 ',' E2 ',' E3# ')' ^genE1E2E3
        minus;
^genE1E2E3 = '(' E1 ',' E2 ',' E3 ')'!.
```

Здесь E1 – универсальное множество U, E2 и E3 – взаимно дополнительные множества A и DA.

Выводы

Базируясь на известных описаниях на Прологе, использующих списковый домен, впервые разработана в метаязыке НФЗ новая практически пригодная формализация *списков, предикатов на списках и множествах*.

Представляется, что составленная метаязыковая формализация практически пригодна не только в качестве новой реализации задач оперирования списками, множествами, но и в качестве спецификации для реализации этих задач в составе других систем.

Литература

1. Братко, Иван. Алгоритмы искусственного интеллекта на языке PROLOG. 3-е издание. Пер. с англ. М.: Издательский дом "Вильямс", 2004. 640 с.
2. Адаменко А.Н., Кучуков А.М. Логическое программирование и Visual Prolog. СПб.: БХВ-Петербург, 2003. 992 с.
3. Клоксин У., Меллиш К. Программирование на языке Пролог. М.: Мир, 1987. 334 с.
4. Abelson, Harold. Structure and interpretation of computer programs. Harold Abelson and Gerald Jay Sussman, with Julie Sussman. 2nd ed. The MIT Press Cambridge, Massachusetts London, England © 1996 by The Massachusetts. 576 p.
5. Haskell 98. Language and Libraries. The Revised Report. Editor by Simon Peyton Jones. Cambridge Academ, 2003. 277 p.
6. Seibel, Peter. Practical Common Lisp. Apress, 2005. 501 p. DOI 10.1007/978-1-4302-0017-8.
7. Кургаев А.Ф., Григорьев С.Н. Нормальные формы знаний. Доп. НАН України. 2015. № 11. С. 36–43.
8. Кургаев А.Ф., Григорьев С.Н. Метаязык нормальных форм знаний. *Кибернетика и системный анализ*. 2016. Том 52. № 6. С. 11–20.
9. Кургаев А.Ф. Формализация списков в метаязыке нормальных форм знаний. Доп. НАН України. 2017. № 10. С. 18–27. doi: https://doi.org/10.15407/dopovidi2017.10.018

References

1. Bratko, Ivan. Prolog Programming for Artificial Intelligence / Third Edition. – Addison-Wesley, 2012. – 696 p.
2. Adamenko A.N., Kuchukov A.M. Logical Programming and Visual Prolog. - St. Petersburg: BHV-Petersburg, 2003. - 992 p. (in Russian).
3. Clocksin, William., Mellish, Christopher S. Programming in Prolog: Using the ISO Standard 5th Edition. – Berlin, Heidelberg: Springer-Verlag, 2003. – 300 p. DOI 10.1007/978-3-642-55481-0
4. Abelson, Harold. Structure and interpretation of computer programs / Harold Abelson and Gerald Jay Sussman, with Julie Sussman. – 2nd ed. / The MIT Press Cambridge, Massachusetts London, England © 1996 by The Massachusetts. – 576 p.
5. Haskell 98. Language and Libraries. The Revised Report / Editor by Simon Peyton Jones. – Cambridge Academ, 2003. – 277 p.
6. Seibel, Peter. Practical Common Lisp. – Apress, 2005. – 501 p. DOI 10.1007/978-1-4302-0017-8
7. Kurgaev, A. The normal forms of knowledge / A.Kurgaev, S.Grygoryev. – Dopov. NAN Ukraine, 2015, № 11. – P. 36-43 (in Russian).
8. Kurgaev, A. Metalanguage of Normal Forms of Knowledge / A.Kurgaev, S.Grygoryev. – Cybernetics and Systems Analysis. – November 2016, 52(6), 839-848. DOI 10.1007/s10559-016-9885-3
9. Kurgaev, A. The Formalization of Lists in the Meta-Language of Normal Forms of Knowledge / Dopov. NAN Ukraine. – 2017. – №10. – P. 18 – 27. (in Russian). doi: https://doi.org/10.15407/dopovidi2017.10.018

Получено 27.01.2020

Об авторе:

Кургаев Александр Филиппович,
доктор технических наук,
профессор, ведущий научный сотрудник.
Количество научных публикаций в украинских изданиях – более 240.
Количество научных публикаций в зарубежных индексированных изданиях – 24.
Индекс Scopus: 2,
h-индекс (Google Scholar): 6
http://orcid.org/0000-0001-5348-2734.

Место работы автора:

Институт кибернетики имени
В.М. Глушкова НАН Украины,
03187, Киев-187,
проспект Академика Глушкова, 40.
Тел.: 8 050 881 62 18.

E-mail: afkurgaev@ukr.net

УДК 004.4'24 https://doi.org/10.15407/pp2020.01.017

А.Ю. Дорошенко, О.А. Яценко

АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ ПРОГРАМ ДЛЯ ПЛАТФОРМИ .NET, ЩО ВИКОРИСТОВУЮТЬ БІБЛІОТЕКУ ПАРАЛЕЛЬНИХ ЗАДАЧ

Виконане налаштування алгебро-алгоритмічного інструментарію на формалізоване проектування та синтез паралельних програм мовою C# для платформи .NET, що використовують засоби бібліотеки паралельних задач TPL. Згадана бібліотека підвищує продуктивність праці розробників за рахунок спрощення процедури додавання паралелізму в програму та динамічно масштабує ступінь паралелізму для найбільш ефективного використання усіх доступних процесорів. В основу пропонованого підходу покладені мова САА-схем, перевагою якої є простота в навчанні й використанні, а також метод конструювання синтаксично правильних програм, що виключає можливість появи синтаксичних помилок у процесі проектування схем. Проведено експеримент з виконання згенерованих за допомогою розробленого інструментарію прикладів паралельних програм на багатоядерному процесорі.

Ключові слова: автоматизоване проектування програм, алгебра алгоритмів, багатопотоковість, бібліотека паралельних задач, паралельні обчислення, синтез програм, TPL.

Вступ

Необхідність у підвищенні продуктивності програмного забезпечення для вирішення трудомістких задач, з одного боку, і нові можливості розпаралелювання обчислень, що надаються багатоядерною архітектурою сучасних мікропроцесорів – з іншого, спонукає до створення спеціалізованих інструментальних засобів для розробки паралельних програм для таких архітектур. Головним способом підвищення продуктивності програм для згаданих платформ є розпаралелювання програм із використанням багатопотоковості. У попередніх роботах [1–5] запропоновані теорія, методологія та інструментарій для автоматизованого проектування, синтезу та перетворення послідовних та паралельних програм, що ґрунтуються на засобах алгебр алгоритмів та переписувальних правил. Зокрема, розглянуті засоби автоматизованої розробки програм для платформи .NET [3–5]; для розпаралелювання програм використовувалися стандартні засоби роботи з потоками.

Одним із способів подальшого підвищення ефективності багатопотокових програм є використання бібліотеки паралельних задач TPL (Task Parallel Library) [6, 7]. Мета TPL полягає у підвищенні продуктивності праці розробників за рахунок спрощення процедури додавання паралелізму в програмні застосунки. TPL динамічно масштабує ступінь паралелізму для найбільш ефективного використання усіх доступних процесорів. Крім того, у бібліотеці паралельних задач здійснюється секціонування роботи, планування потоків у пулі, підтримка відміни, керування станом і виконуються інші низькорівневі задачі. Використовуючи бібліотеку паралельних задач, можна підвищити продуктивність коду, зосередившись на роботі, для якої призначена програма.

У даній роботі запропонований подальший розвиток алгебро-алгоритмічного інструментарію на автоматизоване проектування та генерацію паралельних програм мовою C#, що використовують TPL. Проведено експеримент з виконання згенерованих за допомогою розробленого інструментарію прикладів програм на багатоядерному процесорі.

1. Алгебро-алгоритмічні засоби проектування паралельних програм

Пропонований підхід до конструювання програм ґрунтується на апараті систем алгоритмічних алгебр (САА) та їх модифікацій [1, 2] (САА-М), призначених для формалізації процесів мультиобробки, що виникають при проектуванні програмного забезпечення в мультипроцесорних

системах. САА-М покладено в основу розроблених інструментальних засобів автоматизованого проектування та генерації програм [3–5].

1.1. Системи алгоритмічних алгебр. Модифіковані САА – двоосновна алгебра $GA =< Pr, Op; \Omega_{GA} >$, де Pr – множина логічних умов (предикатів); Op – множина операторів; Ω_{GA} – сигнатура, що складається з логічних операцій (диз'юнкції, кон'юнкції, заперечення) та операторних операцій (композиції, альтернативи, циклу та ін.), що будуть розглянуті далі.

На САА-М грунтується алгоритмічна мова САА/1 [1, 2], призначена для багаторівневого структурного проектування й документування послідовних та паралельних алгоритмів і програм. Перевагою її використання є подання алгоритмів у природно-лінгвістичній формі. Специфікації операторів мовою САА/1 називаються САА-схемами.

Предикати та оператори САА-М поділяються на базисні та складені. Базисні елементи вважаються атомарними, неподільними абстракціями та пов'язані з предметною областю алгоритму, що конструюється.

Складені умови будуються з базисних на основі таких узагальнених булевих операцій [1]:

1) диз'юнкція:

"*condition* 1" OR "*condition* 2";

2) кон'юнкція:

"*condition* 1" AND "*condition* 2";

3) заперечення:

NOT "*condition* 1".

Складені оператори будуються на основі базисних предикатів і операторів та операцій послідовного й паралельного виконання:

1) послідовне виконання операторів (композиція):

"*operator* 1";
"*operator* 2";

2) умовний оператор (альтернатива):

IF '*condition*' THEN
  "*operator* 1"
ELSE
  "*operator* 2"
END IF;

3) цикл типу while:

WHILE '*condition*'
LOOP "*operator*"
END OF LOOP;

4) цикл типу for:

FOR (*counter* FROM *start* TO *fin*)
LOOP "*operator*"
END OF LOOP;

5) паралельне виконання *n* операторів (асинхронна диз'юнкція):

PARALLEL(*i* = 0, ..., *n*)
(
  "*operator i*"
);

6) контрольна точка, яка встановлює значення "істина" для умови '*condition*':

CP '*condition*';

7) синхронізатор, що виконує затримку обчислень доти, поки значення вказаної умови не стане істинним:

WAIT '*condition*',

де '*condition*' може бути пов'язана з умовами, вказаними у контрольних точках.

Приклад 1. Далі як ілюстрація наведено фрагмент послідовної САА-схеми знаходження кількості простих чисел на відрізку [*start, end*]. Схема містить складений оператор "SequentialPrimes (*start, end*)" та умову 'Is prime (*number*)', що відповідають підпрограмам (методам) у цільовій

мові програмування. У схемі виконується послідовна перевірка кожного непарного числа, чи ділиться воно на менші непарні числа. Кількість знайдених чисел зберігається у змінній *result*.

```
"SequentialPrimes(start, end)"
==== "Declare a variable (result)
       of type (long) = (0)";
     FOR (number FROM start TO end – 1)
       IF 'Is prime (number)'
       THEN "Increase (result) by (1)"
       END IF
     END OF LOOP;
     "Return value (result)";

'Is prime (number)'
==== IF (number = 2)
     THEN "Return value (true)"
     END IF;
     IF (number % 2 = 0)
     THEN "Return value (false)"
     END IF;
     "Declare a variable (divisor)
       of type (long) = (3)";
     WHILE NOT (divisor >= number / 2)
     LOOP
       IF (number % divisor = 0)
       THEN "Return value (false)"
       END IF;
       "Increase (divisor) by (2)"
     END OF LOOP;
     "Return value (true)";
```

Приклад 2. Розглянемо далі паралельний варіант алгоритму знаходження кількості простих чисел:

```
"StandardThreadsPrimes(start, end)"
==== "Declare a variable (result)
       of type (long) = (0)";
     "Declare a variable (range)
       of type (long) = (end – start)";
     "Declare a variable (numberOfThreads)
       of type (long)";
      (numberOfThreads :=
       "Get the number of processors");
     "Declare a variable (chunkSize) of type
       (long) = (range / numberOfThreads)";
     PARALLEL(i = 0, ...,
                  numberOfThreads – 1)
      ( "Declare the list of variables
        (chunkStart, chunkEnd)
        of type (long)";
       (chunkStart := start + i * chunkSize);
       IF (i = numberOfThreads – 1)
       THEN (chunkEnd := end)
       ELSE
         (chunkEnd := chunkStart +
          chunkSize)
       END IF;
       FOR (number FROM chunkStart TO
            chunkEnd – 1)
         IF 'Is prime (number)'
         THEN "Increase (result) by (1)
                interlocked"
         END IF
       END OF LOOP;
       CP 'Thread (i) completed work'
      );
     WAIT 'All threads completed work';
     "Return value (result)";
```

Обчислення у даному алгоритмі здійснюються паралельними гілками, кожна з яких обробляє свою порцію даних на ділянці [*chunkStart*, *chunkEnd*], на які порівну розділений відрізок [*start*, *end*]. Кількість потоків дорівнює кількості доступних у системі процесорів. Синхронізація гілок виконується за допомогою контрольних точок та синхронізатора. Сумісно використовуваним потоками ресурсом є змінна *result*. На відміну від послідовного алгоритму, у паралельному для збільшення значення цієї змінної використовується оператор інкременту з блокуванням, а саме: "Increase (*result*) by (1) interlocked".

1.2. Інструментарій автоматизованої розробки програм. Розроблений інтегрований інструментарій проектування та синтезу програм (ІПС) ґрунтується на використанні розглянутих засобів САА-М та методу діалогового конструювання синтаксично правильних програм (ДСП-методу) [1, 2]. На відміну від традиційних синтаксичних аналізаторів, ДСП-метод орієнтований не на пошук і виправлення синтаксичних помилок, а на виключення можливості їх появи в процесі побудови алгоритму. Метод полягає у порівневому конструюванні схем зверху вниз за допо-

могою суперпозиції мовних конструкцій САА-М. На кожному кроці конструювання система надає користувачу на вибір лише ті конструкції, підстановка яких у текст алгоритму, що проектується, не порушує синтаксичну правильність схеми. На основі побудованої схеми алгоритму виконується автоматична генерація тексту програми цільовою мовою програмування. Відображення операцій САА-М у текст мовою програмування подане у вигляді шаблонів і зберігається в базі даних інструментарію. На рис. 1 показано копію екрану системи ІПС із сконструйованим паралельним алгоритмом StandardThreadsPrimes.

Для автоматизації виконання трансформацій алгоритмів система ІПС може застосовуватися спільно з системою TermWare [3–5], що ґрунтується на техніці переписувальних правил.

2. Налаштування алгебро-алгоритмічних засобів на генерацію паралельних програм, що використовують бібліотеку TPL

Бібліотека паралельних задач (TPL) є набором відкритих типів і API-інтерфейсів в просторах імен System.Threading і System.Threading.Tasks в рамках платформи .NET [6]. Ця бібліотека дозволяє автоматично розподіляти навантаження застосувань між доступними процесорами в динамічному режимі, використовуючи пул потоків CLR та конструкції Parallel.For і Parallel.ForEach [7]. Бібліотека TPL виконує розподіл роботи, планування потоків, управління станом та інші низькорівневі задачі. В результаті з'являється можливість максимізувати продуктивність застосувань .NET, не маючи справи із складнощами безпосередньої роботи з потоками.

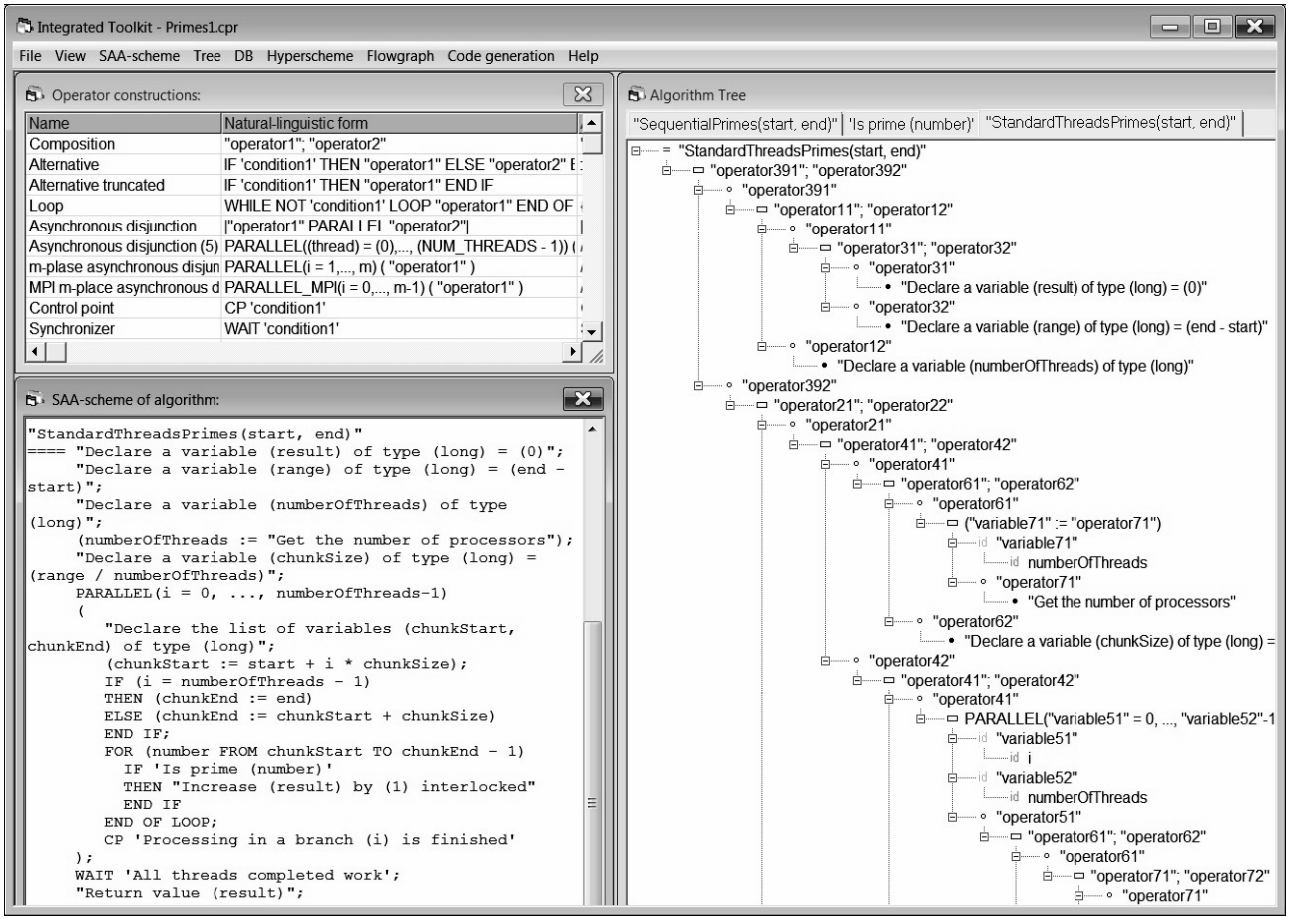


Рис. 1. Копія екрану інструментарію ІПС з побудованою схемою StandardThreadsPrimes

Сутність пулу потоків CLR полягає в наступному. При використанні різних коротких задач, що підлягають виконанню, можна заздалегідь створити набір потоків і потім відправляти відповідні запити, коли настає черга для їх виконання, таким чином, щоб кількість цих потоків автоматично збільшувалася із зростанням необхідності в цих потоках і зменшувалася при виникненні потреби в звільненні ресурсів. Для управління списком потоків в TPL передбачений клас ThreadPool, який в міру необхідності зменшує і збільшує кількість потоків у пулі до максимально допустимого значення. Значення максимально допустимої кількості потоків у пулі може змінюватися. У випадку двоядерного процесора воно за умовчанням становить 1023 робочих потоків і 1000 потоків введення-виведення [8].

Для підтримки використання пулу потоків у САА-М та систему ІПС введено конструкцію, яка відправляє певну операцію "*operator*" (задачу) у чергу на виконання:

```
QUEUE WORK ITEM
(
  "operator"
).
```

Вказана операція виконується, коли стає доступним потік із пулу потоків. Синхронізація потоків виконується за допомогою розглянутих у розділі 1 операцій контрольної точки та синхронізатора.

Приклад 3. САА-схема паралельного алгоритму знаходження кількості простих чисел із використанням пулу потоків є такою:

```
"ThreadPoolPrimes(start, end)"
==== "Declare a variable (result)
       of type (long) = (0)";
     "Declare a variable (chunkSize)
       of type (const long) = (100)";
     "Declare a variable (chunks) of
       type (long)";
     (chunks := (end – start) / chunkSize);
     FOR (i FROM 0 TO chunks – 1)
     "Declare the list of variables
       (chunkStart, chunkEnd)
       of type (long)";
     (chunkStart := start + i * chunkSize);
     IF (i = chunks – 1)
     THEN (chunkEnd := end)
     ELSE (chunkEnd := chunkStart +
           chunkSize)
     END IF;
     QUEUE WORK ITEM
     (
       FOR (number FROM chunkStart TO
            chunkEnd – 1)
         IF 'Is prime (number)'
         THEN "Increase (result) by (1)
               interlocked"
         END IF
       END OF LOOP;
       CP 'Thread (i) completed work'
     )
     END OF LOOP;
     WAIT 'All threads completed work';
     "Return value (result)".
```

У даному алгоритмі вхідний відрізок [*start, end*] поділяється на ділянки розміром *chunkSize* = 100, які обробляються задачами, що заносяться у чергу на виконання і виконуються потоками з пулу.

До алгебро-алгоритмічних засобів було включено також операцію паралельного циклу, яка відповідає конструкції Parallel.For бібліотеки TPL, і ітерації якої розподіляються між доступними у системі процесорами:

```
PARALLEL FOR (counter FROM
              start TO fin)
LOOP "operator"
END OF LOOP
```

Синхронізація потоків у рамках даної операції виконується автоматично.

Приклад 4. САА-схема паралельного алгоритму знаходження простих чисел, що використовує паралельний цикл, наведена далі.

```
"ParallelForPrimes(start, end)"
==== "Declare a variable (result)
       of type (long) = (0)";
     PARALLEL FOR (number FROM
                   start  TO end)
```

```
  IF 'Is prime (number)'
  THEN "Increase (result) by (1)
          interlocked"
  END IF
END OF LOOP;
"Return value (result)".
```

На основі розглянутих у прикладах 1–4 схем в системі ІПС виконана генерація програм мовою C# із використанням бібліотеки TPL. Зокрема, реалізації, що використовують пул потоків та паралельний цикл, є такими:

```
public static long ThreadPoolPrimes(
                    long start, long end)
{
  long result = 0;
  const long chunkSize = 100;
  long chunks;
  chunks = (end - start) / chunkSize;
  var completed = 0;
  var allDone = new
  ManualResetEvent(initialState:
                        false);
  for (long i = 0; i <= chunks - 1; i++)
  {
    long chunkStart, chunkEnd;
    chunkStart = start + i * chunkSize;
    if (i == chunks - 1)
    {
      chunkEnd = end;
    }
    else { chunkEnd = chunkStart +
           chunkSize; };
   ThreadPool.QueueUserWorkItem(
    _ =>
    {
      for (var number = chunkStart;
           number <= chunkEnd - 1;
           ++number)
      {
        if (IsPrime(number))
        {
          Interlocked.Increment(ref
                              result);
        };
      };
      if (Interlocked.Increment(
          ref completed) == chunks)
      {
        allDone.Set();
      }
    });
  };
  allDone.WaitOne();
  return result;
}

public static long ParallelForPrimes(
                    long start, long end)
{
  long result = 0;
  Parallel.For(start, end, number =>
  {
    if (IsPrime(number))
    {
      Interlocked.Increment(ref result);
    };
  });
  return result;
}
```

Результати виконання програм наведено у наступному розділі.

3. Результати експерименту

Проведено експеримент з виконання розроблених програм знаходження простих чисел SequentialPrimes, StandardThreadsPrimes, ThreadPoolPrimes та ParallelForPrimes на 4-ядерному процесорі Intel Core 2 Quad Q9300, 2.5 ГГц. На рис. 2 показано графік залежності часу виконання програм знаходження простих чисел на відрізку $[1, m]$ від значення m. Як видно з графіка, час виконання програм, що використовують TPL, а саме, ParallelForPrimes та ThreadPoolPrimes практично збігається, і є суттєво меншим ніж час виконання програми StandardThreadPrimes, яка застосовує стандартні засоби роботи з потоками.

Максимальне мультипроцесорне прискорення для StandardThreadPrimes становило 2.42 (ефективність використання ядер процесора 0.6), тоді як для програм, що використовують TPL – приблизно 4.0 (ефективність 1.0). Сумарний час виконання програми ThreadPoolPrimes (189.6 секунд) трохи перевищив час виконання ParallelForPrimes (187,9 секунд). Відмітимо також, що реалізація ParallelForPrimes є більш простою та компактною.

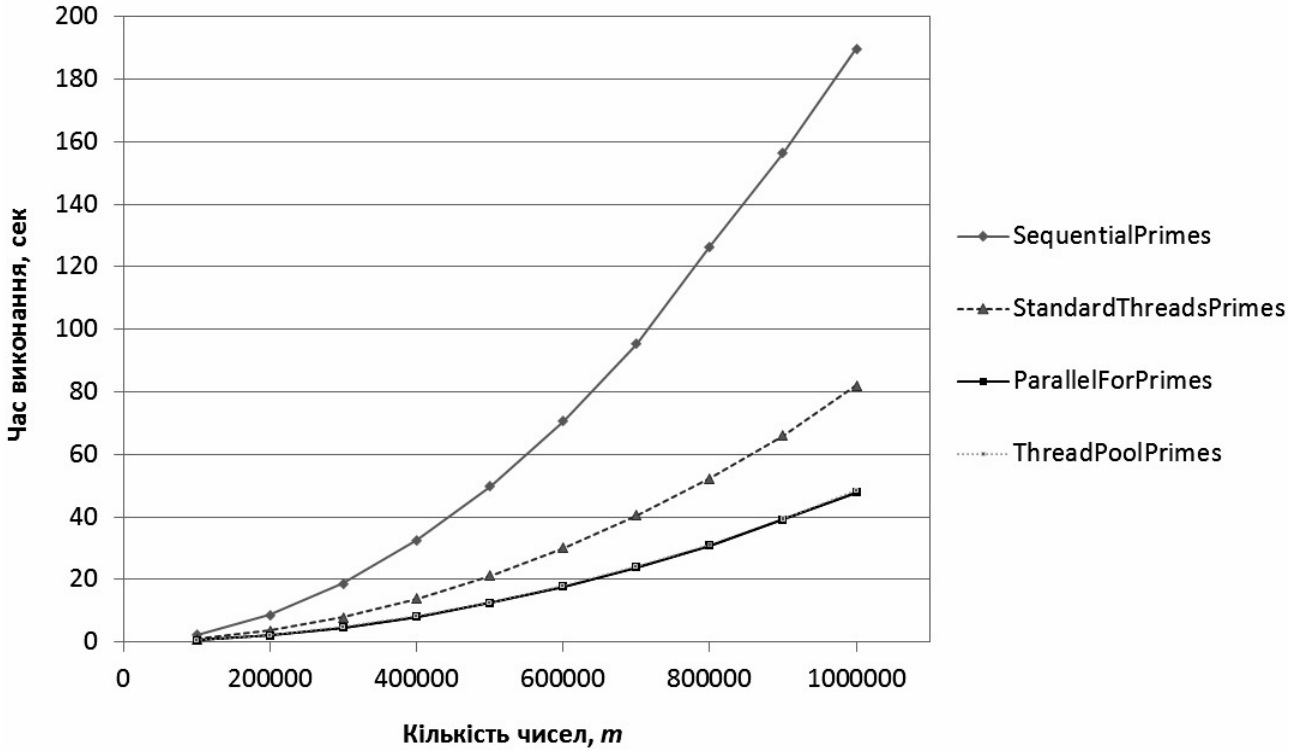


Рис. 2. Залежність часу виконання програм знаходження кількості простих чисел від розміру вхідних даних m

Висновки

Виконане налаштування алгебро-алгоритмічного інструментарію на формалізоване проектування та синтез паралельних програм мовою C#, що використовують засоби бібліотеки паралельних задач TPL. В основу пропонованого підходу покладені мова САА-схем, перевагою якої є простота в навчанні й використанні, та метод конструювання синтаксично правильних програм, що виключає можливість появи синтаксичних помилок у процесі проектування схем.

Проведено експеримент з виконання згенерованих за допомогою системи ІПС паралельних програм знаходження простих чисел на багатоядерному процесорі. Результати експерименту продемонстрували кращий ступінь розпаралелюваності обчислень для програм, що використовують TPL порівняно з програмою, що використовує стандартні засоби програмування потоків.

Література

1. Андон Ф.И., Дорошенко А.Е., Жереб К.А., Шевченко Р.С., Яценко Е.А. Методы алгебраического программирования. Формальные методы разработки параллельных программ. Киев: Наукова думка, 2017. 440 с.
2. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. Алгебро-алгоритмические модели и методы параллельного программирования. Киев: Академпериодика, 2007. 631 с.
3. Жереб К.А. Программный инструментарий, основанный на правилах, для автоматизации разработки приложений на платформе Microsoft .NET. *Управляющие системы и машин*ы. 2009. № 4. С. 51–59.
4. Дорошенко А.Е., Жереб К.А., Яценко Е.А. Об оценке сложности и координации вычислений в многопоточных программах. *Проблеми програмування*. 2007. № 2. С. 41–55.
5. Doroshenko A., Zhereb K., Yatsenko O. Developing and optimizing parallel programs with algebra-algorithmic and term rewriting tools. Communications in Computer and Information Science. Information and Communication Technologies in Education, Research,

and Industrial Applications. Springer, Heidelberg, 2013. Vol. 412. P. 70–92.
6. Task Parallel Library (TPL): сайт. URL: https://docs.microsoft.com/uk-ua/dotnet/standard/parallel-programming/task-parallel-library-tpl (дата звернення: 8.01.2020).
7. Microsoft Task Parallel Library* (TPL): сайт. URL: https://software.intel.com/en-us/node/811328 (дата звернення: 8.01.2020).
8. Васюткина И.А. Разработка клиент-серверных приложений на языке C#. Новосибирск: Изд-во НГТУ, 2016. 112 с.

References

1. Andon, P.I. et al. (2017) *Methods of algebraic programming. Formal methods of parallel program development*. Kyiv: Naukova dumka. (in Russian).
2. Andon, P.I. et al. (2007) *Algebra-algorithmic models and methods of parallel programming*. Kyiv: Akademperiodyka. (in Russian).
3. Zhereb K.A. The rule-based software toolkit for automation of development of applications on Microsoft.NET platform. *Upravl. Sistemy i Mashiny*. 2009. (4). P. 51–59. (in Russian).
4. Doroshenko A.Yu., Zhereb K.A., Yatsenko O.A. On complexity and coordination of computation in multithreaded programs. *Problems in programming*. 2007. (2). P. 41–55. (in Russian).
5. Doroshenko, A., Zhereb, K. & Yatsenko, O. (2013) Developing and optimizing parallel programs with algebra-algorithmic and term rewriting tools. In *Proc. 9th International Conference "ICT in Education, Research, and Industrial Applications" (ICTERI 2013), Revised Selected Papers*. Kherson, Ukraine, 19-22 June 2013. Berlin: Springer. 412. P. 70-92.
6. Docs.microsoft.com. *Task Parallel Library (TPL).* [online] Available from: https://docs.microsoft.com/uk-ua/dotnet/standard/parallel-programming/task-parallel-library-tpl [Accessed 8 Jan. 2020].
7. Software.intel.com. *Microsoft Task Parallel Library* (TPL).* [online] Available from: https://software.intel.com/en-us/node/811328 [Accessed 8 Jan. 2020].
8. Vasyutkina I.A. (2016) *Development of client-server applications in C# language*. Novosibirsk: Publishing House of Novosibirsk State Technical University. (in Russian).

Одержано 11.01.2020

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії комп'ютерних обчислень,
професор кафедри автоматики і управління в технічних системах НТУУ "КПІ імені Ігоря Сікорського".
Кількість наукових публікацій в українських виданнях – понад 150.
Кількість наукових публікацій в зарубіжних виданнях – понад 50.
Індекс Гірша – 5.
http://orcid.org/0000-0002-8435-1451,

Яценко Олена Анатоліївна,
кандидат фізико-математичних наук,
старший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 44.
Кількість наукових публікацій в зарубіжних виданнях – 14.
Індекс Гірша – 2.
http://orcid.org/0000-0002-4700-6704.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559.

E-mail: doroshenkoanatoliy2@gmail.com,
oayat@ukr.net

УДК 004.82 https://doi.org/10.15407/pp2020.01.025

К.А. Кудим, Г.Ю. Проскудина

ОБ ОДНОМ МЕТОДЕ ИЗВЛЕЧЕНИЯ ДАННЫХ ИЗ СЛАБОСТРУКТУРИРОВАННЫХ ДОКУМЕНТОВ

В работе разработан, подробно описан и практически опробован лингвистический метод решения задачи извлечения данных на примере извлечения данных о персоналиях из слабоструктурированных документов, представленных в общедоступном каталоге авторефератов диссертаций Национальной библиотеки Украины им. В.И. Вернадского. Описана вся последовательность шагов: выбор коллекции документов; подготовка документов; написание правил грамматики для извлечения данных из текста; написание правил проверки морфологии; создание интерпретаций или привязок правил к данным; анализ результатов разбора. Лингвистический метод извлечения выявил ряд преимуществ по сравнению с описанным ранее методом извлечения данных с помощью регулярных выражений.

Ключевые слова: слабоструктурированные документы, извлечение информации, лингвистический анализатор, синтаксический анализатор, морфологический анализ, контекстно-свободная грамматика.

Введение

Ранее нами был описан метод извлечения данных из слабоструктурированных текстов с помощью регулярных выражений [1]. В настоящей работе метод усовершенствован заменой регулярных выражений на полноценный синтаксический анализатор. А также существенно улучшен метод предварительной подготовки документов.

В работе разработан, подробно описан и практически апробирован лингвистический метод решения задачи извлечения данных на примере извлечения данных о персоналиях из авторефератов диссертаций. Описана вся последовательность шагов: выбор коллекции документов; подготовка документов; написание правил грамматики для извлечения данных из текста; написание правил проверки морфологии; создание интерпретаций или привязок правил к данным; анализ результатов разбора. Лингвистический метод извлечения выявил ряд преимуществ по сравнению с описанным ранее методом извлечения данных с помощью регулярных выражений.

1. Коллекция документов

Для разработки и тестирования метода необходимо прежде всего определиться с коллекцией документов. В качестве типового документа было решено остановиться на авторефератe диссертации. Такой документ имеет ряд необходимых признаков слабоструктурированного документа. В частности, в документе содержится большое количество метаданных о нём самом. Это не только имя автора и название, а также организации, научные руководители и оппоненты, тема диссертации, научное звание и специальность.

В качестве тестовой коллекции авторефератов диссертаций для исследования была выбрана коллекция полнотекстовых документов электронного каталога авторефератов диссертаций Национальной библиотеки Украины им. В.И. Вернадского. В корпусе авторефератов диссертаций представлено более чем 63000 документов. Коллекция состоит из всех авторефератов поданных в Украине за период 1998 – 2011 гг. Полные тексты авторефератов содержатся в файлах преимущественно в форматах RTF и DOC. Коллекция доступна через навигацию по каталогу в виде ссылок в метаописании каждого автореферата. Для обработки на локальной машине была создана программа-краулер, которая обходит электронный каталог используя навигационные ссылки, и скачивает файлы содержащие полный текст. В результате вся коллекция доступна для дальнейшей обработки в локальном каталоге.

2. Подготовка документа

Перед непосредственным извлечением данных исходные файлы обрабатывались в два этапа.

На первом шаге подготовки документа из файлов различных форматов извлекается текст. С помощью готовых утилит операционной системы документы, такие как DOC и RTF, легко преобразуются в простой неструктурированный текстовый формат TXT. Такой же подход можно обнаружить в зрелых программных продуктах для систем электронных библиотек EPrints и DSpace, которые используют подобное преобразование с помощью внешних программ для полнотекстовой индексации документов сложных для анализа форматов. В настоящей работе использовалась утилита textutil из операционной системы macOS. Пример консольной команды для преобразования файла показан на рис. 1.

```
textutil -convert txt file.doc
```

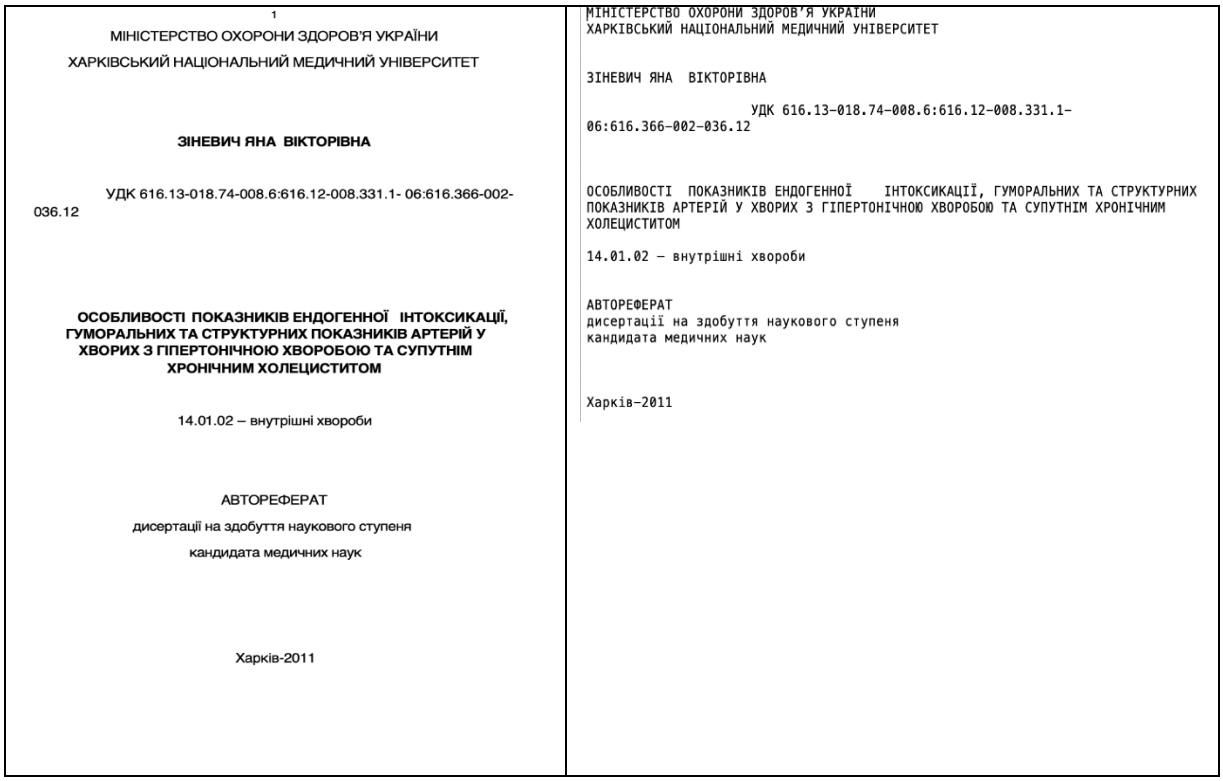
Рис. 1. Пример использования утилиты *textutil* для преобразования формата документа

На втором шаге подготовки документов, для того, чтобы ускорить синтаксический анализ, работающий недостаточно быстро для обработки целого документа, извлекается только титульная страница автореферата. Такой подход оказался допустимым, поскольку основная часть документа не содержит интересующих нас метаданных об автореферате. Эта информация находится на титульной странице либо на первых двух страницах автореферата. Титульную страницу извлекали из уже преобразованного простого текстового файла. Опытным путём было установлено, что 50 первых строк достаточно для извлечения всей необходимой информации и для существенного сокращения времени дальнейшей обработки. На рис. 2 показан пример команды для извлечения начальных строк из текстового файла.

```
head -n 50 file.txt > head.txt
```

Рис. 2. Пример использования утилиты *head* для усечения документа

На выходе после предварительной подготовки документа получаем файл, содержащий 50 строк текста, о которых известно, что в них записана титульная страница автореферата диссертации (см. рис. 3). Дальнейшая задача состоит в том, чтобы из этого небольшого текстового файла извлечь метаданные диссертации.



1

МІНІСТЕРСТВО ОХОРОНИ ЗДОРОВ'Я УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ МЕДИЧНИЙ УНІВЕРСИТЕТ

ЗІНЕВИЧ ЯНА ВІКТОРІВНА

УДК 616.13-018.74-008.6:616.12-008.331.1- 06:616.366-002-036.12

ОСОБЛИВОСТІ ПОКАЗНИКІВ ЕНДОГЕННОЇ ІНТОКСИКАЦІЇ, ГУМОРАЛЬНИХ ТА СТРУКТУРНИХ ПОКАЗНИКІВ АРТЕРІЙ У ХВОРИХ З ГІПЕРТОНІЧНОЮ ХВОРОБОЮ ТА СУПУТНІМ ХРОНІЧНИМ ХОЛЕЦИСТИТОМ

14.01.02 – внутрішні хвороби

АВТОРЕФЕРАТ

дисертації на здобуття наукового ступеня

кандидата медичних наук

Харків-2011

МІНІСТЕРСТВО ОХОРОНИ ЗДОРОВ'Я УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ МЕДИЧНИЙ УНІВЕРСИТЕТ

ЗІНЕВИЧ ЯНА ВІКТОРІВНА

УДК 616.13-018.74-008.6:616.12-008.331.1-06:616.366-002-036.12

ОСОБЛИВОСТІ ПОКАЗНИКІВ ЕНДОГЕННОЇ ІНТОКСИКАЦІЇ, ГУМОРАЛЬНИХ ТА СТРУКТУРНИХ ПОКАЗНИКІВ АРТЕРІЙ У ХВОРИХ З ГІПЕРТОНІЧНОЮ ХВОРОБОЮ ТА СУПУТНІМ ХРОНІЧНИМ ХОЛЕЦИСТИТОМ

14.01.02 – внутрішні хвороби

АВТОРЕФЕРАТ
дисертації на здобуття наукового ступеня
кандидата медичних наук

Харків-2011

Рис. 3. Титульный лист автореферата до и после предварительной подготовки

Минимальный набор возможных полей метаданных подлежащих извлечению:

- автор,
- тема,
- научная степень.
- раздел науки.

Дополнительные поля могут быть следующие:

- научный руководитель,
- оппоненты,
- организация,
- и другие.

3. Извлечение данных

Ранее нами уже был описан метод извлечения данных из слабоструктурированных текстов с помощью регулярных выражений [1]. В настоящей работе метод усовершенствован заменой регулярных выражений на полноценный синтаксический анализатор.

Известные существующие системы для лингвистического анализа произвольных русскоязычных текстов от Yandex[1] и Pullenti[2] представляют собой чёрные ящики с закрытым исходным кодом [2]. По этой причине они сложны в настройке и накладывают ограничения на программное окружение.

Сравнительно недавно был разработан лингвистический анализатор Yargy [3] с открытым исходным кодом на языке Python. Он позволяет выполнять синтаксический разбор текста на русском языке, что критически важно для наших целей, поскольку для английского языка существует множество решений, перенастройка которых на русский язык по сложности сравнима с написанием собственного лингвистического анализатора.

Анализатор Yargy изначально настроен на русский и, частично, украинский языки. Наша дополнительная задача доработать Yargy таким образом, чтобы он полноценно выполнял синтаксический разбор текстов на украинском языке. Эта задача упрощается тем, что в основе библиотеки Yargy есть встроенный морфологический модуль, который уже понимает, как русский, так и украинский язык. Благодаря этому модулю анализ украиноязычных текстов возможен «из коробки», однако одной лишь морфологии часто бывает недостаточно. Иногда происходят ошибки определения частей речи, иногда неправильно выделяются части слова. Например, мужское имя «Виктор» (рус.) и «Віктор» (укр.). В русском языке библиотека правильно определяет слово целиком как имя человека. В украинском языке выделяется приставка «вік» и корень «тор», тем самым препятствуя правильной классификации слова и затрудняя правильный анализ документа.

Рассмотрим библиотеку Yargy детальнее. Реализована библиотека целиком на языке Python, что позволяет запускать её везде, где можно использовать интерпретатор Python. На вход библиотека принимает, во-первых, специально подготовленную контекстно-свободную грамматику, настроенную на определённый тип документа в зависимости от задачи, и, во-вторых, собственно текст для анализа. На выходе работы анализатора получаем граф синтаксического разбора входного текста по данной грамматике. Таким образом, библиотека не содержит встроенную грамматику, что делает анализатор очень гибким и пригодным для разнообразных типов задач. Доступны два режима работы с входным текстом:

- найти все совпадения с грамматикой в тексте, то есть все подстроки, которые распознаются данной грамматикой;
- проверить, распознаётся ли весь заданный текст грамматикой.

Сама грамматика описывается также на чистом Python, что удобно, поскольку не требует установки различных сред и изучения дополнительного внутреннего языка. И хотя для описания грамматики авторами Yargy был разработан специальный язык DSL (domain specific language), он все же является подмножеством языка Python.

Синтаксический разбор реализован с помощью алгоритма Эрли [4]. Алгоритм

[1] https://yandex.ru/dev/tomita/
[2] www.pullenti.ru

кубической сложности, позволяющий разбирать произвольные тексты по контекстно-свободной грамматике, которая не требует приведения к нормальной первой форме Хомского. Контекстно-свободная грамматика, заданная на входе алгоритма, может содержать неоднозначности, что не будет препятствовать её разбору этим алгоритмом.

Дополнительным преимуществом при использовании Yargy является возможность добавить статистические анализаторы. Библиотеке на вход можно подать размеченные тексты, в которых лингвистические признаки слов такие как род, число, падеж, часть речи выписаны в последовательности вероятностных правил, что за чем следует и с какой вероятностью. Далее если в новом анализируемом тексте встречается незнакомое слово, которого нет в словарях, то библиотека может, основываясь на лингвистических признаках цепочки предыдущих слов, спрогнозировать род, падеж, число для незнакомого слова. Результаты такого рода обучения можно использовать как дополнительный вход библиотеки Yargy. В рамках настоящей работы статистический синтаксический анализ подробно не рассматривается.

Кроме синтаксического разбора библиотека Yargy позволяет размечать каждое слово в соответствии с его морфологией, то есть получать результаты морфологического анализа. Приписывание каждому слову род, число, падеж и часть речи помогает более детально настраивать дерево разбора. Используемый морфологический модуль pymorphy2 позволяет привести форму слова к нормальному виду (т. е. единственное число, именительный падеж для имен существительных, инфинитив для глаголов), что позволяет одновременно выполнять преобразования извлекаемых данных для сохранения сущностей в удобной форме. Например, если имя, фамилия и отчество человека встретилось в тексте в родительном падеже, то в базе данных они сохраняются в именительном падеже.

Модуль pymorphy2 основывает морфологический анализ на размеченном корпусе текстов русского языка OpenCorpora[3]. Поэтому морфологический анализ украинских слов не работает должным образом.

4. Грамматика

Для дальнейшей обработки подготовленных документов была построена грамматика, распознающая титульный лист автореферата диссертации.

Правила. На рис. 4 показан пример написания правил в этой грамматике.

```
NAME = rule(
   gram('NOUN')
)
PERSON = rule(
   SURN,
   NAME,
   PATR
)
```

Рис. 4. Фрагмент правила грамматики для извлечения имени из текста

В каждом автореферате диссертации присутствует автор. В приведенном примере правила указана фамилия SURN, имя NAME и отчество PATR. Именно в таком порядке. Каждое из этих слов — имя существительное. В грамматике указывается какой частью речи должно быть имя автора. В приведенном фрагменте имя NAME это имя существительное NOUN. В целом данное правило гласит, что если встречается подряд три существительных, то это полное имя человека, в данном случае имя автора автореферата. Это имя позднее будет извлечено для сохранения в базе и связи его с другими извлечёнными сущностями.

Также написаны отдельные правила для фамилии и отчества, описывающие дополнительные признаки для каждого части имени. Правила охватывают всевозможные сочетания для имён, но достаточно узко, чтобы не захватывать лишние данные. С помощью библиотеки синтаксического анализа Yargy построенная контекстно-свободная грамматика, применяющаяся к неструктурированному тексту

[3] http://opencorpora.org/

для получения на выходе структурированных данных, как в рассмотренном примере фамилию, имя и отчество автора. Если правила написаны достаточно хорошо, то эти данные определяются во входном документе. В противном случае мы получаем либо ложное срабатывание, когда грамматика либо её часть срабатывает не на тех данных, которые предполагалось извлечь с помощью этих правил, либо документ не соответствует ни одному варианту структуры документа, описанному грамматикой, и тогда распознавания не происходит вообще.

Для выделения имени человека существуют уже написанные более сложные правила. В частности, над рассматриваемой библиотекой Yargy[4] реализована использующая её библиотека Natasha[5], предназначенная для извлечения имен, адресов, организаций и некоторых других сущностей. Наборы правил из Natasha можно легко использовать в своих грамматиках для Yargy [3]. Для нашего корпуса текста грамматика распознавания русских имён из Natasha была доработана, поскольку в изначальном виде распознавание было успешным лишь на малом количестве тестовых документов.

Последовательность обработки текста при синтаксическом анализе следующая. Прежде всего весь текст разбивается на токены. Эта часть алгоритма предельно проста. Пробелы игнорируются, любая последовательность букв или цифр считается токеном, а также любой знак препинания тоже считается токеном. Такой подход позволяет писать правила, использующие знаки препинания для распознавания. Так можно распознавать специальные отформатированные данные, такие как УДК или даты.

Предикаты. В правилах можно использовать предикаты. Правило, показанное на рис. 4, использует предикат о том, что слово является существительным. На рис. 5 приводится правило для базового распознавания предметной классификации УДК.

Поле данных УДК представляет собой само слово “УДК”, за которым следуют цифры, точки, дефисы, двоеточия. В приведённом правиле используется два предиката eq и type для проверки полного совпадения и для проверки совпадения типа токена с целым числом либо со знаком препинания. Правило действует таким образом: сначала проверяется точное совпадение со словом “УДК”, затем с помощью логической связки or (логическое “или”) проверяется совпадение по одному из двух предикатов, что, либо тип токена это число, либо тип токена – знак пунктуации. Также можно указать, что правило является повторяемым, указав ключевое слово repeatable() для части правила.

```
UDK = rule(
   eq('УДК'),
   or_(
      type('INT'),
      type('PUNCT')
   ).repeatable()
)
```

Рис. 5. Фрагмент правила, распознающего предметную классификацию УДК

Морфология. Существует возможность проверять совпадения с любой формой проверяемого слова. Такая проверка, например, для слов 'кандидат' и 'доктор' осуществляется с помощью правила, показанного на рис. 6. Здесь идет проверка совпадения слов кандидата, кандидату, кандидаты, доктора, доктору и др. Для этого нужно либо выписать все словоформы, либо написать правило о том, чтобы находить совпадения в любом падеже, числе и роде. Либо использовать библиотечную функцию, упрощающую эту рутинную морфологическую работу.

```
DEGREE = rule(morph pipeline([
   'кандидат',
   'доктор'
]))
```

Рис. 6. Правило проверки морфологии слов 'кандидат' и 'доктор'

[4] https://github.com/natasha/yargy
[5] https://github.com/natasha/natasha

5. Интерпретация

После того как сделан разбор, и получено дерево синтаксического разбора текста, возникает задача привязки узлов дерева разбора к структурам данных. При использовании библиотеки Yargy для этой цели создаются объекты Python. Выше мы рассматривали пример простой грамматики для разбора полного имени человека. Для этой грамматики можно использовать объект Person на языке Python, показан на рис. 7. После анализа объект будет конкретизирован данными из результатов синтаксического разбора.

Привязка объекта к грамматике осуществляется с помощью так называемой интерпретации. К правилу грамматики приписывается объект, с помощью которого будет происходить интерпретация результатов работы этого правила. На рис. 8 показана реализация интерпретации правила распознавания имени объектом Person.

```
Person = fact(
   'Person',
   ['surn', 'name', 'patr']
)
```

Рис. 7. Объект Person

```
PERSON = rule(
   SURN,
   NAME,
   PATR
). interpretation(
   Person
)
```

Рис. 8. Интерпретация правила или привязка объекта к правилу

После составления дерева разбора, когда парсер распознает соответствующий текст, каждому интерпретируемому узлу дерева будет сопоставлен объект и распознанными текстовыми значениями будут инициализированы поля этого объекта. Каждое правило будет узлом либо листом в дереве разбора. Составные правила, которые включают в себя другие правила, будут узлом в дереве. Терминальные же символы и предикаты будут листьями такого дерева. Интерпретация узла будет объектом, а интерпретация листа будет атрибутом объекта. Например, значение имени будет атрибутом объекта Person.name.

6. Результаты разбора

На рис. 9 в формате JSON показан полный результат разбора титульной страницы автореферата.


```
{
  "author": {
    "surn": "Зіневич",
    "name": "Яна",
    "patr": "Вікторівна"
  },
  "udk": "УДК 616.13-018.74-008.6:616.12-
008.331.1- 06:616.366-002-036.12",
  "title": "ОСОБЛИВОСТІ ПОКАЗНИКІВ ЕН-
ДОГЕННОЇ ІНТОКСИКАЦІЇ, ГУМОРАЛЬ-
НИХ ТА СТРУКТУРНИХ ПОКАЗНИКІВ АР-
ТЕРІЙ У ХВОРИХ З ГІПЕРТОНІЧНОЮ
ХВОРОБОЮ ТА СУПУТНІМ ХРОНІЧНИМ
ХОЛЕЦИСТИТОМ",
  "specnum": "14.01.02",
  "specname": "внутрішні хвороби",
  "degree": "кандидат",
  "degreespec": "медичних наук",
  "city": "Харків",
  "year": "2011"
}
```


Рис. 9. Результат разбора титульной страницы автореферата в формате JSON

Далее на рис. 10 показана часть графа дерева разбора для Имени автора. Объект автор диссертации – Thesis.author, это персона Person, фамилия персоны Person.surn – ЗАЛОЗНОВА, имя персоны Person.narne – Юлія, отечество персоны Person.patr – Станіславівна. Это дерево разбора уже с привязанными объектами.

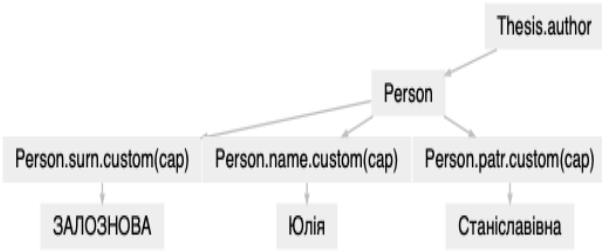


Рис. 10. Фрагмент дерева разбора

7. Дальнейшая работа

Сохранение и отображение данных. Следующая задача состоит в том, чтобы сохранить извлечённые результаты в базе данных для того, чтобы можно было делать запросы и представлять извлечённую информацию публично в удобочитаемом виде. Для хранения данных мы отдаём предпочтение формату RDF, поскольку в нём естественным образом поддерживается неограниченное количество атрибутов и связей, можно обрабатывать автоматически и есть возможность строить SPARQL-запросы.

Улучшение разбора. Предстоит существенно улучшить украинскую морфологию, а также добавить словари имен. Такие словари можно получить извлечением из названий статей Википедии, поскольку там уже есть большое количество имён, фамилий и отчеств, причём формат известен заранее. Извлечь их можно из соответствующих страниц категорий Википедии, где имена представлены списками на одной странице. Благодаря тому, что формат известен, можно разделить название статьи на имя, фамилию и отчество, и сохранить каждое поле в отдельный файл, получая таким образом словари имён, фамилий и отчеств. Эти словари необходимы для более точного распознавания имён. Если слово найдено в словаре, то распознавание такого слова в тексте становится тривиальной задачей с большой вероятностью успешного результата.

Выводы

В работе пошагово и на многочисленных примерах описан метод извлечения данных из слабоструктурированных текстов корпуса авторефератов диссертаций с помощью использования синтаксического анализатора Yargy.

В дальнейшем предстоит улучшить украинскую морфологию и добавить словари. А также разработать схему хранения извлеченных данных с тем, чтобы их можно было бы публично использовать и строить к ним различные запросы.

Работа выполняется в рамках темы "Методы и средства создания интеллектуальных сервис-ориентированных информационно-обеспечивающих систем в среде семантического Веба".

Литература

1. Кудим К.А., Проскудина Г.Ю. Методы и средства извлечения данных о персоналиях из авторефератов диссертаций. *Проблеми програмування*. 2019. № 2. С. 38–46.
2. Рубайло А.В., Косенко М.Ю. Программные средства извлечения информации из текстов на естественном языке. *Альманах современной науки и образования.* № 12 (114). 2016. С. 87–92. http://scjournal.ru/articles/issn_1993-5552_2016_12_23.pdf
3. Кукушкин А. Наташа – библиотека для извлечения структурированной информации из текстов на русском языке. https://habr.com/ru/post/349864/
4. Earley J. An efficient context-free parsing algorithm, Communications of the Association for Computing Machinery. 13:2:94-102. 1970.

References

1. Kudim K.A., Proskudina G.YU. (2019). Methods and tools for extracting personal data from theses abstracts *Problems in programming*. [online – pp.isofts.kiev.ua] (2). P. 38–46. (in Russian). Available from: http://pp.isofts.kiev.ua/ojs1/article/view/359 [Accessed 6/05/2019]. DOI: https://doi.org/10.15407/pp2019.02.038
2. Rubailo A.V., Kosenko M.Y. Software tools for information extraction from natural-language texts. *Almanac of modern science and education*. № 12 (114) 2016. P. 87–92. (in Russian). http://scjournal.ru/articles/issn_1993-5552_2016_12_23.pdf
3. Kukushkin A. Natasha – a library for extracting structured information from texts in Russian. (in Russian). https://habr.com/ru/post/349864/
4. Earley J. An efficient context-free parsing algorithm, Communications of the Association for Computing Machinery, 13:2:94-102, 1970.

Получено 03.02.2020

***Об авторах*:**

Кудим Кузьма Алексеевич,
младший научный сотрудник.
Количество научных публикаций в украинских изданиях – 18.
Количество научных публикаций в зарубежных изданиях – 2.
http://orcid.org/0000-0001-9483-5495,

Проскудина Галина Юрьевна,
научный сотрудник.
Количество научных публикаций в украинских изданиях – 31.
Количество научных публикаций в зарубежных изданиях – 15.
http://orcid.org/0000-0001-9094-1565.

Место работы авторов:

Институт программных систем
НАН Украины,
03187, Киев-187,
проспект Академика Глушкова, 40.
Тел.: (044) 526 6033.

E-mail: kuzmaka@gmail.com,
guproskudina@gmail.com

УДК 681.3 https://doi.org/10.15407/pp2020.01.033

І.Ю. Гришанова, Ю.В. Рогушина

РОЗРОБКА МЕТОДІВ КЕРУВАННЯ ДОСТУПОМ ДО ІНФОРМАЦІЇ WIKI-РЕСУРСАХ

В роботі аналізуються технологічні засади розробки Wiki-ресурсів на основі програмного забезпечення MediaWiki, розглядаються проблеми доступу до інформації у цих ресурсах та наводяться методи та засоби вирішення цих проблем. Значна увага приділяється архітектурі MediaWiki та складовим цієї архітектури. Розроблені в результаті аналізу рішення апробовано у розробці порталу Великої української енциклопедії (е-ВУЕ) .

Ключові слова: MediaWiki, Wiki-ресурс, керування доступом, Велика українська енциклопедія.

Вступ

Wiki-технології є основою для створення структурованих інформаційних ресурсів великого обсягу, орієнтованих на одночасне використання через Web великою кількістю користувачів. Таке подання інформації є зручним та інтуїтивно зрозумілим для широких кіл користувачів, має достатню виразну здатність для відображення досить складних змістових відношень між окремими елементами ресурсу.

Нині Wiki-технології широко застосовуються для створення різноманітних відкритих енциклопедійних ресурсів. Наприклад, портальна версія Великої української енциклопедії (е-ВУЕ) базується на програмному забезпеченні MediaWiki [1], дозволяє подавати авторський та рецензований мультимедійний контент із складною системою зв'язків та навігації [2]. Процес створення контенту є централізованим та контролюється модераторами наукових напрямків, що виключає безпосередню участь користувачів у редагуванні даних. Частина даних у е-ВУЕ призначена для забезпечення семантичного пошуку та навігації і не орієнтована на безпосереднє використання кінцевими користувачами, що призводить до необхідності керувати доступом до інформації відповідно до ролей користувачів цього порталу. Але технологія MediaWiki не підтримує безпосередньо такий функціонал, і тому виникає потреба у розробці таких спеціалізованих програмних засобів, що дозволяють вирішувати ці задачі.

Wiki-технологія створення відкритих інформаційних ресурсів

Wiki-ресурси являють собою перспективний шлях розвитку приватних і публічних баз знань. Оригінальна система Wiki була винайдена в 1995 р. У. Каніngемом [3] для Web-вузла Pattern Languages Community, щоб спростити спільне створення й документування програмного забезпечення. Всесвітньо відомий приклад застосування технології Wiki – це Вікіпедія, найбільша з безкоштовних онлайн-енциклопедій [4].

Під *Wiki-технологією* зазвичай розуміють таку технологію побудови Wiki-ресурсу, яка дає змогу відвідувачам брати участь у редагуванні його вмісту – виправляти помилки, додавати нові матеріали, не використовуючи спеціальні програми, не реєструючись на сайті й не вивчаючи мову HTML [5].

Кожна окрема сторінка Wiki-ресурсу називається **Wiki-сторінка** (*Wiki page*). Створення й редагування Wiki-сторінок не потребує від користувачів знання мови HTML: механізм Wiki надає можливість написання документів за допомогою простої мови розмітки й Web-браузеру. Всі Wiki-сторінки мають унікальні назви, на які можна посилатися з інших статей, і вміст. Інформація, представлена у Wiki, має нелінійну навігаційну структуру: кожна сторінка, зазвичай, містить велику кількість гіперпосилань на інші сторінки.

У 2006 році термін Wiki додано до онлайнового Оксфордського Словника Англійської мови (OxfordEnglish Dictionary, OED) як позначення моделі сайтів, контент яких може змінювати сам користувач.

Формат Wiki-сторінок (Wiki-текст) – це спрощена мова розмітки, що використовується для того, щоб виділити в тексті різні структурні й візуальні елементи або вказати на них. Кожна Wiki-система має власний стиль і синтаксис залежно від реалізації. У багатьох реалізаціях Wiki гіперпосилання показується таким, яким воно є насправді, на відміну від HTML, де невидиме гіперпосилання може мати довільний видимий текст або зображення.

Для створення та підтримки Wiki-ресурсу необхідно спеціальне програмне забезпечення (ПЗ) – *Wiki-рушій***,** який забезпечує роботу відповідного Web-сайту. Рушій Wiki-сайту робить запис змін так, щоб у будь-який час сторінку можна було б повернути до кожного з її попередніх станів. Wiki-система може також містити різні інструментальні засоби для простого контролювання стану Wiki, що постійно змінюється, а також для обговорювання й розв'язання проблем, що виникають через різні погляди на зміст Wiki-сторінок.

Популярність технологій Wiki сприяла появі великої кількості реалізацій практично для всіх можливих платформ і конфігурацій ПЗ. Крім того, різні реалізації Wiki неоднаково розширюють основні можливості технології, додаючи, наприклад, убудовані графічні редактори чи сервіси WAP. Але загальна орієнтованість на створення відкритих інформаційних ресурсів (ІР) призводить до певних проблем, що пов'язані з обробкою персональних даних та із захистом інформації від несанкціонованого доступу (приміром, до службової інформації, що не призначається для кінцевого користувача, але необхідна для ефективної організації Wiki-ресурсу).

MediaWiki як технологічна основа побудови Wiki-ресурсів

Широко вживаним рішенням для створення Wiki-ресурсів є MediaWiki [6]. Це програмне забезпечення розроблено для створення найбільш відомого Wiki-ресурсу – Вікіпедії. Воно враховує розподілену розробку незалежними розробниками додаткових плагінів та функціоналу без зміни архітектури.

MediaWiki розроблено для підтримки спільноти, яка створює та керує вільне повторно використовуване знання на відкритій платформі. Тому MediaWiki не включає стандартний функціонал корпоративних систем керування контентом (CMS), натомість пропонує різноманітні інструменти для боротьби зі спамом та вандалізмом.

Архітектура MediaWiki повинна враховувати не тільки розподілене використання ПЗ, постійно зростаючу кількість користувачів та авторів контенту, патрулювання для збереження цілісності контенту, боротьби зі спамом та вандалізмом, а також і розподілений процес розробки ПЗ як афілійованими розробниками, так і окремими сторонніми користувачами, надавши їм можливість додавати власні додатки до загальної бази програмного коду.

Для організації спільного відкритого процесу розробки з використанням зусиль різних розробників та волонтерів, було обрано такі найбільш поширені та оптимальні інструменти: LAMP платформа – мова PHP (з подальшим використанням фреймворків Symphony та інших, як Zend Framework), база даних MySQL/MariaDB (використовується як базова з можливістю зміни на PostgreSQL, SQLite тощо), jQuery як клієнтська бібліотека JavaScript. Розробка відбувається в стилі відкритого коду [7] і координується спеціальними спільнотами та групами WikiMedia Foundation. Документація коду генерується автоматично, загальна інформація публікується на сторінках Вікіпедії.

В ретроспективі визнано, що багато рішень прийняті не зовсім оптимально, а деякі були помилковими. Але зважаючи на невелику кількість розробників і волонтерський характер участі у проекті, швидкість розробки і невеликий обсяг коду, важко критикувати засновників проекту. Проект зростає, розвивається і постійно викорис-

товується, тому базові рішення (навіть ті, які нині визнано не оптимальними) здебільшого кардинально не змінюються, а ведеться пошук рішень, які зберігають існуючу архітектуру.

На даний час нам не вдалося знайти більш-менш сталого і змістовного визначення архітектури MediaWiki. Ресурси, пов'язані із розширенням функціоналу MediaWiki, містять базові рекомендації для розробників, опис змінних, опис таблиць БД, але не надають формалізований опис архітектури. Для забезпечення безпеки, розробники ядра і ревізіонери коду розробили жорсткі правила кодування, а також надали набори обгорток (wrappers) для екранування вхідних html-даних та ескейпування sql-запитів (класи WebRequest, Sanitizer).

MediaWiki може мати багато можливих варіацій конфігурацій, параметри яких зберігаються в глобальних php-змінних. Значення за замовчуванням встановлені в DefaultSettings.php, але адміністратор може змінити їх, вказавши нові або додаткові значення в файлі LocalSettings.php. З точки зору безпеки та можливості виникнення конфліктів, це не є добрим рішенням, але процес розвитку ПЗ MediaWiki має довгу і ще не закінчену історію руху від глобальних змінних до об`єктів. Для збереження наступності версій і підтримки старих, конфігурація зберігається за допомогою змінних.

MediaWiki використовує реляційну БД (за замовчуванням це MySQL), і Вікіпедія використовує саме цю БД, але спільнота розробила підтримку інших БД, таких як PostgreSQL, Oracle, SQLite. Під час первинної процедури встановлення адміністратор обирає необхідну БД. ПЗ MediaWiki забезпечує рівень абстракції БД і рівень абстракції запитів до БД, що полегшує розробникам доступ до неї.

В процесі розвитку MediaWiki схема БД багаторазово змінювалася, найбільш значною зміною було розділення місця зберігання текстів і відслідкування ревізій в MediaWiki 1.5. На даний час БД складається з десятків таблиць, які містять дані про контент (наприклад page, revision, category, recentchanges), про користувачів (user, user_groups), про медіа файли (image, filearchive), кешування (objectcache, l10n_cache, querycache), внутрішні засоби (job – для виконання черги задач), а також може містити інші додаткові дані, що встановлені додатково та є необхідними для певних плагінів.

Абстрактна модель архітектури MediaWiki

Архітектура MediaWiki на абстрактному рівні містить (рис. 1):

- базу даних;
- базове ядро MediaWiki;
- набір плагінів, які розширюють функціональні можливості ПЗ;
- набір скінів (шаблонів відображення контенту сторінок та засобів навігації).

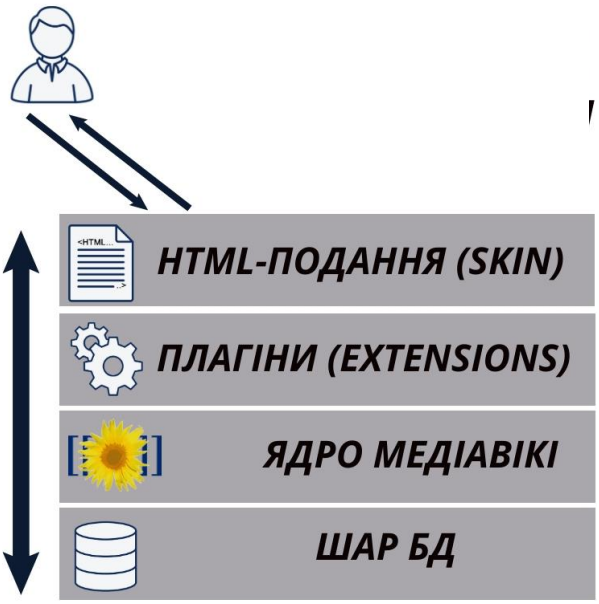


Рис. 1. Абстрактна модель архітектури MediaWiki

Архітектурно відображення контенту Web-сторінок (навігації та контенту сторінок) максимально відокремлено відповідно до MVC моделі (model-view-control) і винесено на рівень скінів (шаблонів відображення), хоча деякі невеликі частини інтерфейсу, які історично створені раніше, не вдалось повністю відокремити від основного коду бізнес-логіки.

Набір стилів оформлення (так звані skin) дозволяють користувачам модифікувати зовнішнє оформлення MediaWiki [8]. Зазвичай базовий комплект програмного забезпечення містить 4 базових набора: Vector, Monobook, Cologne Blue та Modern. Стиль, встановлюваний за замовчуванням – Vector. Деякі набори стилів не підтримують мобільної версії. Додаткові компоненти, такі як JavaScript, також функціонують тільки в певних наборах стилів.

На даний час в e-BYE прийнятий для використання як базовий додатковий набір стилів оформлення Foreground, який фокусується на виділенні контенту, підтримує адаптивні макети та має спеціальні заздалегідь визначені класи для підтримки функціоналу Semantic MediaWiki. Цей набір побудовано на Zurb's Foundation Framework (v4.3.2), який орієнтований у першу чергу на підтримку мобільних пристроїв та розширеного адаптивного фронт-енд фреймворку.

Скін (skin) – це набір елементів, що визначають дизайн інтерфейсу (стилі, меню, шрифти тощо). У MediaWiki скіни – це PHP-класи, кожен з яких розширює батьківський клас Skin; вони містять функції, які збирають інформацію, необхідну для генерування HTML. На відміну від плагінів, розробка або налаштування скінів – тяжка і нестандартна задача. Наприклад, стандартний скін "MonoBook" довгий час було важко налаштувати під нову версію, оскільки він містив багато стилів CSS, які підтримують старі браузери; редагування шаблону або CSS вимагало багатьох наступних змін, щоб відобразити зміни для всіх браузерів і платформ.

Скін задає стиль оформлення сторінок (з використанням CSS та JavaScript):

- розташування елементів навігації, інформаційних блоків, кнопок тощо;
- фон сторінок та фон елементів сторінки;
- типи та розміри шрифтів для різних типів абзаців;
- розміри відступів від блоків та елементів сторінки.

Набір стилів оформлення розроблюється і задається один раз і поширюється на всі сторінки.

Зазвичай Web-сторінка може мати стандартний набір зон: навігаційний блок зверху, зони меню навігації ліворуч та праворуч, основний зміст сторінки в середині, і внизу сторінки – зона підвалу. Набір стилів оформлення Foreground повністю змінює макет зовнішнього відображення Wiki-сторінки та дозволяє додавати нові елементи інтерфейсу. В e-BYE всі функціональні меню перенесені в горизонтальний навігаційний рядок у вигляді випадаючих меню, що знаходиться наверху сторінки.

Стандартна сторінка MediaWiki за замовчуванням містить певний обмежений набір компонентів (рис. 2):

- елементи персональних інструментів та налаштування для зареєстрованого користувача;
- меню набору дій, які можливо робити з сторінкою (редагувати, обговорення, перегляд історії тощо);
- форма пошуку по сайту;
- системні повідомлення (site notice);
- гасло сторінки;
- рядок сторінки, вищої за таксономічним поділом та джерело перепосилання (redirected from);
- блок контенту сторінки, який може містити текст, мультимедійну інформацію, формули, таблиці, підзаголовки, блок змісту. Контент сторінки може розділятися на частини, кожна частина може редагуватися окремо, для чого з`являється елемент «Редагувати» над кожною виділеною частиною;
- блок категорій, до яких належить сторінка;
- нижній колонтитул з піктограмами та посиланнями;
- ліва бічна панель (sidebar), яка містить посилання на певні головні сторінки, сторінки довідок, блок посилань на інструменти та блок посилань на версії сторінки іншими мовами;
- індикатор статусу сторінки.

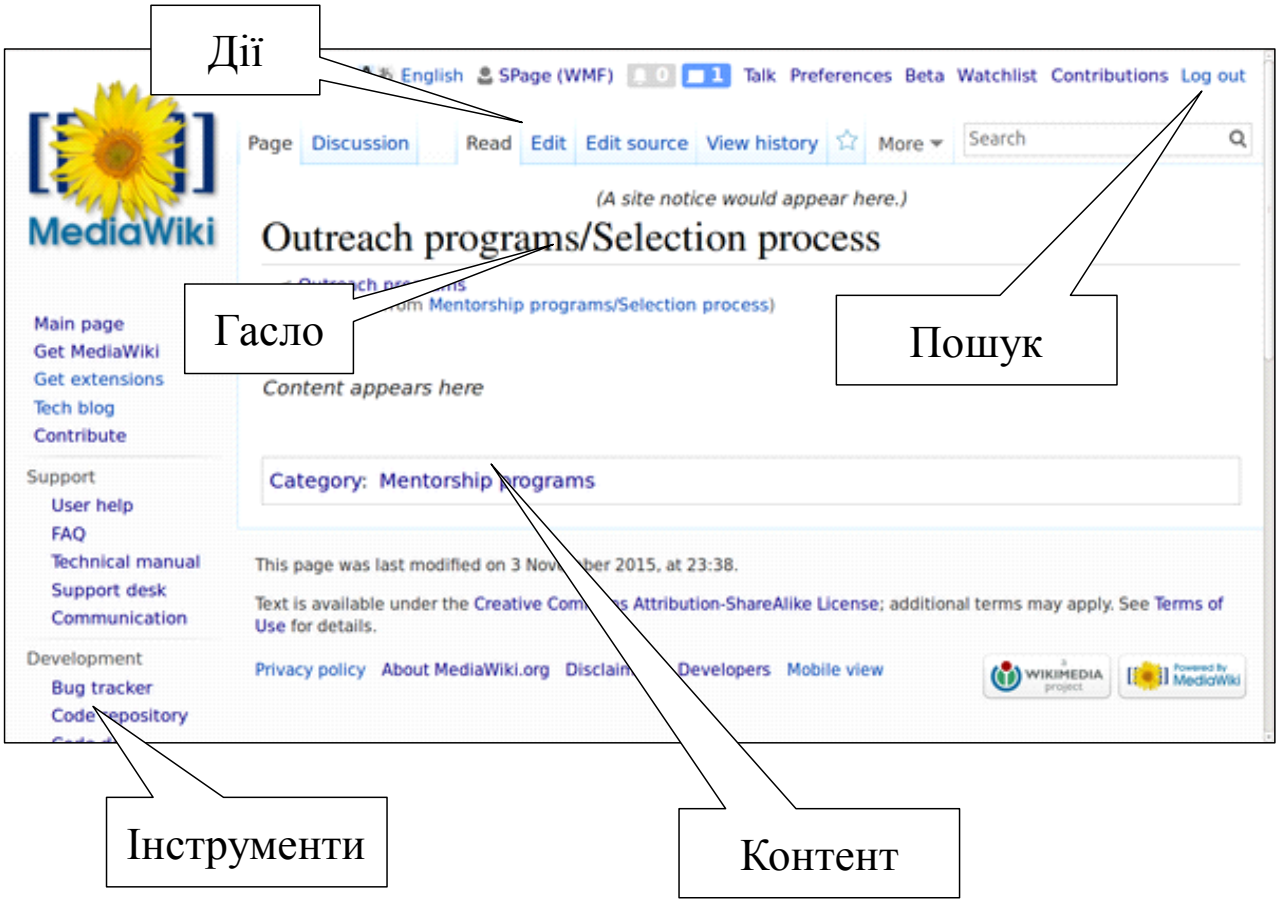


Рис. 2. Основні елементи Wiki-сторінки

Процес виконання Web-запиту в MediaWiki

Головною точкою доступу в MediaWiki є index.php, який керує більшістю запитів, поданих до серверу. Код цього модулю виконує перевірку безпеки, завантажує встановлені конфігураційні значення з includes/DefaultSettings.php, виконує конфігурацію за допомогою includes/Setup.php, після чого встановлює конфігураційні змінні, що відповідають конкретному серверу і прописані в файлі LocalSettings.php. Потім він створює об'єкт MediaWiki ($mediawiki) та створює об'єкт Title ($wgTitle) в залежності від заголовка та параметрів дії, вказаних в URL-запиті.

index.php може приймати різні параметри дії, вказані в URL-запиті; дією за замовчуванням є перегляд (*view*), яка показує звичайний вигляд вмісту статті. Наприклад, запит https://en.wikipedia.org/w/index.php?title=Apple&action=view відображає зміст статті "Apple" в англійській Вікіпедії. Також часто застосовують наступні дії: *edit* (відкриття статті для редагування), *submit* (для попереднього перегляду редагування або збереження статті), *history* (для відображення історії змін статті) та *watch* (для додавання статті до списку спостереження користувача). Адміністративні дії включають *delete* (для видалення статті) та *protect* (захист для запобігання редагуванню статті).

Для обробки більшості URL-запитів викликається метод MediaWiki::performRequest(). Він перевіряє наявність поганих заголовків, обмеження читання, локальні переспрямування та цикли переспрямування та визначає для якого типу сторінки запит – для звичайної чи спеціальної.

Запити звичайної сторінки передаються методу MediaWiki::InitializeArticle(), який створює для сторінки об’єкт Article ($wgArticle), а потім методу MediaWiki::performAction(), який обробляє "стандартні" дії. Після завершення дії виконується MediaWiki::finalCleanup(), який

доопрацьовує запит, здійснюючи закінчення транзакцій БД, виводячи згенерований HTML і запускаючи відкладені оновлення, які поставлені в чергу завдань. MediaWiki::restInPeace() закінчує виконання відкладених оновлень та закриває завдання.

Якщо запитувана сторінка є спеціальною (тобто не звичайна сторінка Wiki-ресурсу, а спеціалізована сторінка, пов'язана з відповідним програмним забезпеченням або виконанням певних спеціальних операцій, наприклад, Статистика), замість InitializeArticle() викликається SpecialPageFactory::ExecutePath і виконується відповідний PHP скрипт. Спеціальні сторінки можуть робити всілякі операції і кожна має певну мету, як правило, незалежну від будь-якої статті чи її змісту. Спеціальні сторінки включають різного роду звіти (останні зміни, журнали, категорії без категорій) та засоби адміністрування Wiki (блоки користувачів, зміни прав користувача). Послідовність їх виконання залежить від їх функції.

Багато функцій містять код профілювання, який дає можливість, якщо профілювання включено, слідкувати за робочим процесом виконання, що корисно для відлагодження. Профілювання здійснюється за допомогою виклику функцій *wfProfileIn* та *wfProfileOut*, які відповідно запускають та зупиняють профілювання; обидві функції приймають ім'я функції як параметр. На сайтах WikiMedia (таких як Вікіпедія, Вікідата тощо), щоб зберегти продуктивність, профілювання зазвичай відключене або проводиться на один відсоток від усіх запитів.

Під час перегляду сторінки HTML-код може бути взято з кешу. Якщо ж кешування відключене або сторінка кожен раз генерується заново, то спочатку розгортаються шаблони, функції парсеру та змінні. В результаті генерується розширений Wiki-текст, що є проміжним результатом, який можна побачити за допомогою *Special: ExpandTemplates*, і який залежить від:

- Wiki-тексту;
- шаблонів, на які прямо чи опосередковано є посилання;
- функцій і параметрів парсеру, на які прямо чи опосередковано є посилання;
- значень змінних, на які прямо чи опосередковано є посилання.

Далі цей розширений Wiki-текст перетворюється в HTML-код; він надсилається користувачеві та містить посилання на CSS, JavaScript та файли зображень. Користувач може побачити цей проміжний результат, застосувавши в браузері опцію «view source». HTML-код для певної сторінки залежить від:

- розширеного Wiki-тексту;
- режиму, перегляд чи редагування (див. далі);
- наявності внутрішньо пов'язаних сторінок (дає посилання для перегляду чи редагування);
- скіну та інших налаштувань користувача;
- імені користувача;
- статусу користувача (більше посилань, якщо адміністратор тощо);
- простору імен (визначає посилання на сторінку Talk або у випадку Talk-сторінки – відповідну сторінку);
- чи відслідковує користувач сторінку (надає посилання для включення чи відключення відслідковування за змінами);
- чи була нещодавно відредагована Talk-сторінка користувача (видає повідомлення).

Нарешті, браузер рендерить HTML з використанням файлів, на які є посилання. Результат, який користувач бачить на екрані, залежить від:

- HTML-коду;
- файлів, на які посилається код HTML, таких як вставлені зображення, CSS-файли на стороні сервера та файли JavaScript;
- браузера та його налаштувань, включаючи, можливо, локальний файл CSS та роздільну здатність екрана.

Якщо JavaScript реагує на подію, таку як клацання миші, сторінка на екрані також залежить від цих подій. Це може виникати, наприклад, у разі виконання сортування таблиці.

Якщо користувач обирає вкладку “Редагувати”, клієнту надсилається тільки Wiki-текст (контент сторінки або окремого розділу цієї сторінки з Wiki-розміткою, але без навігаційних елементів). Якщо користувач натискає «Попередній перегляд», на сервер для обробки надсилається нова версія Wiki-тексту, і після обробки сервер надсилає нову версію HTML-коду. Якщо користувач натискає кнопку "Зберегти", надсилаючи нову версію статті на сервер, система записує здійснені редагування та надсилає браузеру HTML-код нової версії (знову). У деяких випадках на цьому етапі також відбувається автоматичне перетворення Wiki-тексту.

Кешування в MediaWiki

Оскільки MediaWiki є базовим ПЗ усіх сайтів WikiMedia, він максимально оптимізований для реалізації високої продуктивності.

На сайтах WikiMedia більшість запитів обробляються зворотним кешуванням проксі-серверів (Squids) і ніколи не роблять це на серверах застосунків MediaWiki. Проксі-сервери містять статичні версії цілих відтворених сторінок, які служать для простого читання для незареєстрованих користувачів. MediaWiki спочатку підтримує Squid і Varnish та інтегрується з цим шаром кешування, наприклад, сповіщаючи їх про очищення сторінки з кеша, коли вона була змінена. Для зареєстрованих користувачів та виконання інших запитів, які не можуть обслуговувати проксі, Squid пересилає запити на Web-сервер (Apache).

Другий рівень кешування використовується, коли MediaWiki надає та збирає сторінку з декількох об'єктів, багато з яких можна кешувати, щоб мінімізувати майбутні звернення. Такі об'єкти включають інтерфейс сторінки (бічна панель, меню, текст інтерфейсу користувача) та власне вміст, проаналізований з Wiki-тексту. Кеш об'єктів у пам'яті доступний в MediaWiki з ранньої версії 1.1 (2003), і це особливо важливо, тому що дозволяє уникнути повторного розбору сторінок із складною структурою та великим обсягом.

Дані сесій також можуть зберігатися у *memcached*. Починаючи з версії 1.16, MediaWiki використовує виділений кеш-об'єкт для локалізованого тексту інтерфейсу; це було додано після того, як помітили, що значна частина об'єктів, кешованих у пам'яті, складалася з повідомлень інтерфейсу, поданих мовою користувача.

Останній шар кешування складається з кешування коду PHP (PHP opcode cache), яке зазвичай використовується для прискорення роботи PHP програм. Компіляція може бути тривалим процесом; щоб уникнути компіляції PHP-скриптів у код виконання кожного разу, коли вони викликаються, для зберігання зібраного коду і виконання його безпосередньо без компіляції може використовуватися PHP-акселератор. MediaWiki може працювати з багатьма акселераторами, такими як APC, PHP-акселератор та eAccelerator.

Також шар кешування абстрактних об'єктів MediaWiki дозволяє зберігати кешовані об’єкти в декількох місцях, включаючи файлову систему, базу даних або opcode cache.

Структура контенту MediaWiki

Простори імен (namespaces), так само як і спеціальні Wiki-сторінки, були вперше використані в PHP-скрипті для Нупедії (попередниці Вікіпедії) розробником MediaWiki Г.М. Манське [9]. Впродовж багатьох років namespaces застосовувалися у Wiki-ресурсах, виконуючи лише одну функцію – розділяти різні види контента. За написом namespaces є префіксом, який відокремлений від назви сторінки двокрапкою (наприклад, *Talk*:, *File*: та *Template*:); в основному просторі імен вмісту префікса немає. Простори імен виявились важливою особливістю MediaWiki, оскільки вони створюють необхідні передумови для спільноти Wiki та встановлюють дискусії на метарівні, процеси спільноти, портали, профілі користувачів тощо.

Простори імен дозволяють структурувати контент Wiki-ресурсів, розділити контент за функціональним призначенням. В межах одного простору імен сторінки можуть бути впорядковані за темами за допомогою категорій. Існує певний список

вже зарезервованих просторів імен: *User* – для розділення користувачів, *Mediawiki* – для службових сторінок ядра MediaWiki, *File* – для зберігання інформації про файли, *Special* – для службових сторінок, *Template* – для сторінок шаблонів.

Як було зазначено вище, в загальному вигляді Web-серверна архітектура розподіляє процес отримання результату на етапи: серверну обробку запиту, підготовку відповіді, відправку та обробку отриманих даних на клієнті і відображення їх в браузері (або іншому клієнтському ПЗ).

Відповідно до цього підходу, процес обробки сервером запиту користувача складається з наступних етапів:

- виконання запиту користувача і підготовка необхідних для відсилання даних;
- перетворення даних у кінцевий формат шляхом трансформації та обгортання в html-формат, додавання елементів навігації, інтерфейсу та дизайну;
- відправлення згенерованої Web-сторінки клієнту (браузеру).

Архітектура MediaWiki пропонує різні способи налаштування та розширення програмного забезпечення. Це можна зробити на різних рівнях доступу:

- системні адміністратори можуть встановлювати розширення, скіни, налаштовувати окремі допоміжні програми wiki (наприклад, для відображення мініатюр для зображень та генерування TeX) та глобальні налаштування;
- користувачі Wiki з правами sysops (іноді їх також називають "адміністраторами") можуть редагувати гаджети, налаштування JavaScript та CSS;
- будь-який зареєстрований користувач може налаштувати власний інтерфейс, використовуючи свої уподобання (для існуючих налаштувань, скінів та гаджетів) або внести зміни до власної моделі інтерфейсу (використовуючи свої особисті сторінки JS та CSS).

Зовнішні програми також можуть спілкуватися з MediaWiki через його API, якщо він включений, в основному роблячи будь-яку функцію та дані доступними для користувача.

MediaWiki забезпечує систему гачків (hooks) – перехвату, що дозволяють стороннім розробникам запускати користувацький PHP-код до, після або замість коду MediaWiki для конкретних подій. Розширення MediaWiki використовують такі гачки для підключення свого коду.

До появи в MediaWiki цих засобів перехвату процес додавання спеціального коду PHP викликав зміну основного коду. За допомогою гачків можливо розширити мову Wiki-розмітки додатковими можливостями, вводячи нові теги.

Екосистема MediaWiki дозволяє розробникам створювати плагіни, які називаються розширеннями (extensions) і використовують гачки. Система розширень має низку недоліків: реєстрація розширень базується на виконанні додаткового коду при кожному запуску системи, а не на кешованих даних, що обмежує абстрагування та оптимізацію та шкодить продуктивності MediaWiki. Але в цілому сучасна архітектура розширень є досить гнучкою інфраструктурою, яка допомагає зробити спеціалізований код більш модульним, утримуючи основне програмне забезпечення від занадто великого збільшення та дозволяючи стороннім користувачам будувати власну функціональність на основі стандартної версії MediaWiki.

Використання авторизації для визначення прав доступу до інформації

В MediaWiki реалізовано можливість розподілу функціоналу за допомогою авторизації. В момент встановлення Wiki-системи є можливість одразу визначити, закритий чи відкритий тип матиме ця система. Якщо система встановлюється як відкрита, то кожен незареєстрований користувач має повний доступ до контенту всіх сторінок.

Авторизація – керування рівнями та засобами доступу до певного захищеного ресурсу, як у фізичному розумінні (доступ до кімнати готелю за карткою), так і в галузі цифрових технологій (наприклад, автоматизована система контролю доступу) та ресурсів системи залежно від ідентифікатора і пароля користувача або надання

певних повноважень (особі, програмі) на виконання деяких дій у системі обробки даних [10].

З позицій інформаційної безпеки авторизація – частина процедури надання доступу для роботи в інформаційній системі, після ідентифікації і автентифікації.

Авторизація контролює доступ легальних користувачів до ресурсів системи після успішного проходження ними автентифікації. Найчастіше процедури автентифікації й авторизації поєднують. За допомогою авторизації встановлюються й реалізуються права доступу до ресурсів.

Створення облікового запису користувача у MediaWiki означає, що цей користувач вказує ім'я облікового запису (логін) і пароль. Для наступного входу користувачу потрібно вказати логін і знову підтвердити пароль.

Для перегляду будь-якого відкритого Wiki-ресурсу на основі MediaWiki авторизація не потрібна. Як правило, вона не потрібна й для того, щоб редагувати контент. Проте доступ з авторизацією надає користувачу кілька важливих переваг. Інші користувачі зможуть довідатися про діяльність користувача (зміни, внесені у сторінки) за логіном користувача. Це зручніше, ніж використання для цього IP-адреси користувача, яка може змінюватися, якщо користувач заходить до Wiki-ресурсу з різних пристроїв та місць. Спрощується спілкування з іншими користувачами Wiki-ресурсу, простіше відслідковувати зміни в цікавлячих сторінках, використовуючи список спостереження, та виникає можливість отримувати автоматичні повідомлення про важливі події.

Незважаючи на те, що технологія Wiki орієнтована на розробку відкритих джерел інформації, в процесі функціювання таких ресурсів виникає потреба у захисті деяких фрагментів даних від несанкціонованого доступу. Зазвичай виникає необхідність закривати деякі певні сторінки, які містять конфіденційну бізнес-інформацію або є частиною робочого процесу установи, що розробляє відповідний Wiki-ресурс.

Це викликано як потребою у захисті персональних даних, так і доцільністю закриття технічних деталей реалізації від кінцевих користувачів.

Зважаючи на відкриту ідеологію Wiki-систем, відповідні механізми в базовій версії MediaWiki відсутні. Існуючі плагіни, розроблені для цього сторонніми розробниками або мають лише бета-версію, або взагалі вже не підтримуються і є застарілими. Тому необхідно, використовуючи існуючі технологічні засоби середовища MediaWiki, створити механізм, який забезпечить захист елементів контенту Wiki-ресурсу для різних груп користувачів цього ресурсу. Додатковою умовою є максимальна незалежність такого рішення від оновлення версій ПЗ, гнучкість і простота його використання.

Постановка задачі

Враховуючи існуючу архітектуру системи і принцип відкритості Wiki-ресурсу, необхідно створити технічне рішення – надбудову над системою, яка не змінить кардинально код системи, буде логічним і простим у використанні, але дозволить диференціювати доступ до інформації для різних категорій користувачів відкритого Wiki-ресурсу відповідно до їх ролей, повноважень та інформаційних потреб.

Для вирішення такої задачі пропонується: виконати семантичний поділ контенту (сторінок, що складають Wiki-ресурс) на окремі множини, що не перетинаються, відповідно до їх функцій та типу вмісту, використовувати механізм розподілу контенту за просторами імен, який існує в WikiMedia; розробити механізм перехвату інформації, що згенерована на сервері системою, яка буде враховувати знання про цей розподіл та пов’язані з ним правила обробки для того, щоб передавати на клієнт тільки ті відомості, які є дозволеними для цієї групи користувачів.

Алгоритм керування доступом до контенту Wiki-ресурсу

У найбільш узагальненому розумінні, для рішення проблеми керування доступу до контенту містить такі підзадачі:

- розподіл користувачів на групи доступу відповідно до їх прав доступу до окремих елементів контенту (наприклад, на зареєстрованих і незареєстрованих);
- розподіл контенту на групи (з використанням таких механізмів MediaWiki, як простори імен та категорії, а також семантичного розширення Semantic MediaWiki – семантичних властивостей сторінок та їх значень);
- створення шаблонів типових інформаційних об'єктів (ТІО), що описують ті дані, доступ до яких має бути диференційованим для користувачів з різних груп доступу;
- розробка правил співставлення належності користувача до певної групи доступу з тим, яку саме інформацію він має право отримувати;
- реалізація механізму для отримання набору даних, що доступні для конкретного користувача, що надає можливість аналізувати згенерований контент перед відправкою його на клієнт;
- створення шаблонів для подання інформації відповідно до аналізу, що виконано у попередньому пункті.

Розглянемо окремі випадки реалізації цього підходу. Для таких випадків пропонується використовувати механізм розподілу контенту за просторами імен, що існує в WikiMedia, і розробити механізм перехоплення інформації, що згенерована на сервері системою, яка буде передавати на клієнт тільки те, що дозволено.

У загальному випадку з урахуванням специфіки MediaWiki, що проаналізована вище, вхідними даними задачі є:

- розподіл користувачів на зареєстрованих і незареєстрованих;
- поділ контенту на різні простори імен та категорії (та можливість додавати до сторінок певні шаблони);
- можливість додавати програмний код, здатний аналізувати контент, згенерований MediaWiki, перед відправкою даних на клієнт.

Архітектура Wiki побудована таким чином, що на останньому етапі перед відправкою згенерованої сторінки вона оброблюється на рівні скіна. На цьому етапі можливо додати до інформації певні html-теги, а також зробити більш глибокий аналіз контенту. Найбільш простим є аналіз простору імен, до якого відноситься сторінка, що аналізується.

Скін Foreground програмно реалізовано двома класами: class Skinforeground, який розширює базовий клас ядра SkinTemplate і клас формату подання – class foregroundTemplate, який розширює BaseTemplate. Разом такий формат відображення і весь програмний код скінів подано в файлі типу Назва Скіну.skin.php. [11]. Розробка власного скіну надає можливість вносити такі операції аналізу, не впливаючи на ядро MediaWiki. Внаслідок цього виникає можливість аналізувати контент відповідно до набору обраних умов та згенерувати новий контент відповідно до власних потреб.

Узагальнений алгоритм роботи такого скіну містить наступні етапи:

- формування сторінки контентом згідно отриманих параметрів ядром Wiki в змінній $*body*;
- аналіз змісту змінної $*body* і внесення змін відповідно до прав користувача і правил доступу;
- відображення змінної $*body* в браузері.

Перевірка, чи зареєстрований користувач, здійснюється методом – *$isLoggedIn = $wgUser->isLoggedIn().*

Отримання простору імен поточної сторінки, здійснюється методом: *$namespace = $this->getSkin()->getTitle()->getNsText();* Інші параметри сторінки отримуються відповідно.

Виконання перевірки та отримання дозволу виконується функцією *$allow_to_show = check_allow_to_show_content ($isLoggedIn, $namespace, $isArticle, $title, $content).*

Ця функція працює за наступною логікою:

if ($allow_to_show == 1),

зміна контенту згідно з правилами показу

else

зміна контенту згідно з правилами.

Функція визначення дозволу має наступний вигляд:

```
function
check_allow_to_show_content($isLoggedIn,
$namespace, $isArticle, $title, $content) {
        $allow_to_show = 1;
        if ($isLoggedIn != 1) {
        if($namespace=="XXXX")
                        $allow_to_show
= 0;
        elseif ($isArticle) {
              $allow_to_show          =
check_page_edition_status($vue_content);
              }
        }
        return $allow_to_show;
       }
```

Функція *check_page_edition_status ($content)* аналізує контент статті (сторінки) і визначає дозвіл на показ чи редагування поточної сторінки для певного користувача.

Таким чином, підключення аналізатору до скіну дозволяє проаналізувати сторінку і надати користувачеві тільки ту інформацію, яку можна показувати саме йому. Таке рішення не змінює основного ядра ПЗ, не залежить від версії ПЗ і дозволяє безболісно встановлювати оновлення ядра і додаткових плагінів.

Можливість MediaWiki семантично розділяти контент шляхом використання механізму категорій та просторів імен надає багато інших можливостей і переваг для створення енциклопедичного контенту та корпоративних інформаційних ресурсів, тому що це забезпечує не тільки відображення контенту, але й інтеграцію цієї системи в робочий процес підприємства. Запропоноване рішення надає більше можливостей для використання MediaWiki не тільки як базу знань, але й як допоміжну довідкову систему для керування внутрішніх корпоративних бізнес-процесів документообігу.

Важливо зазначити, що запропоноване рішення не стосується захисту доступу до API MediaWiki та від різних типів хакерських атак тощо.

Використання методу керування доступом в е-ВУЕ

Запропонований вище підхід використовується для розширення функціоналу портальної версії Великої української енциклопедії (е-ВУЕ). Наразі структура БЗ е-ВУЕ формально зафіксована за допомогою відповідної Wiki-онтології, яка є результатом співпраці експертів ПрО (у даному випадку – модераторів галузей знань, що представлені в енциклопедії) з інженерами із знань. Для складної структури знань, що характерна для е-ВУЕ, це дозволяє значно чіткіше описувати знання та запобігати повторного використання імен категорій з різними значеннями [12] та імпортувати знання з інших онтологій [13].

Проаналізувавши структуру БЗ, ми виділили наступні простори імен, які потребують спеціалізованої обробки з точки зору доступу до параметрів сторінок:

- шаблон;
- подія;
- артефакт;
- зображення;
- медіафайли;
- аудіосупровід;
- модератор.

Кожен з цих просторів імен має специфічне призначення в е-ВУЕ і має використовуватися тільки розробниками ресурсу у службових цілях. Наприклад, простір імен Артефакт забезпечує виведення на першу сторінку порталу відомостей про ті артефакти, створення яких пов’язано з поточною датою.

Інший приклад – шаблони виведення інформації про типові елементи контенту е-ВУЕ. Доступ до таких шаблонів мають отримувати тільки розробники цього ресурсу, автори та модератори наукових напрямків (рис. 3).

Цей шаблон “Персоналія” призначений для опису типових інформаційних об’єктів, пов’язаних з конкретними особами, – персоналій. За допомогою цього шаблону забезпечується уніфіковане подання інформації про осіб на тих сторінках Wiki-ресурсу, що їм відповідають.

Користувачі е-ВУЕ, що не авторизовані, не отримують доступ до цього шаблону. Тому контент таких сторінок буде схованим від них за допомогою розглянутих вище засобів (рис. 4).

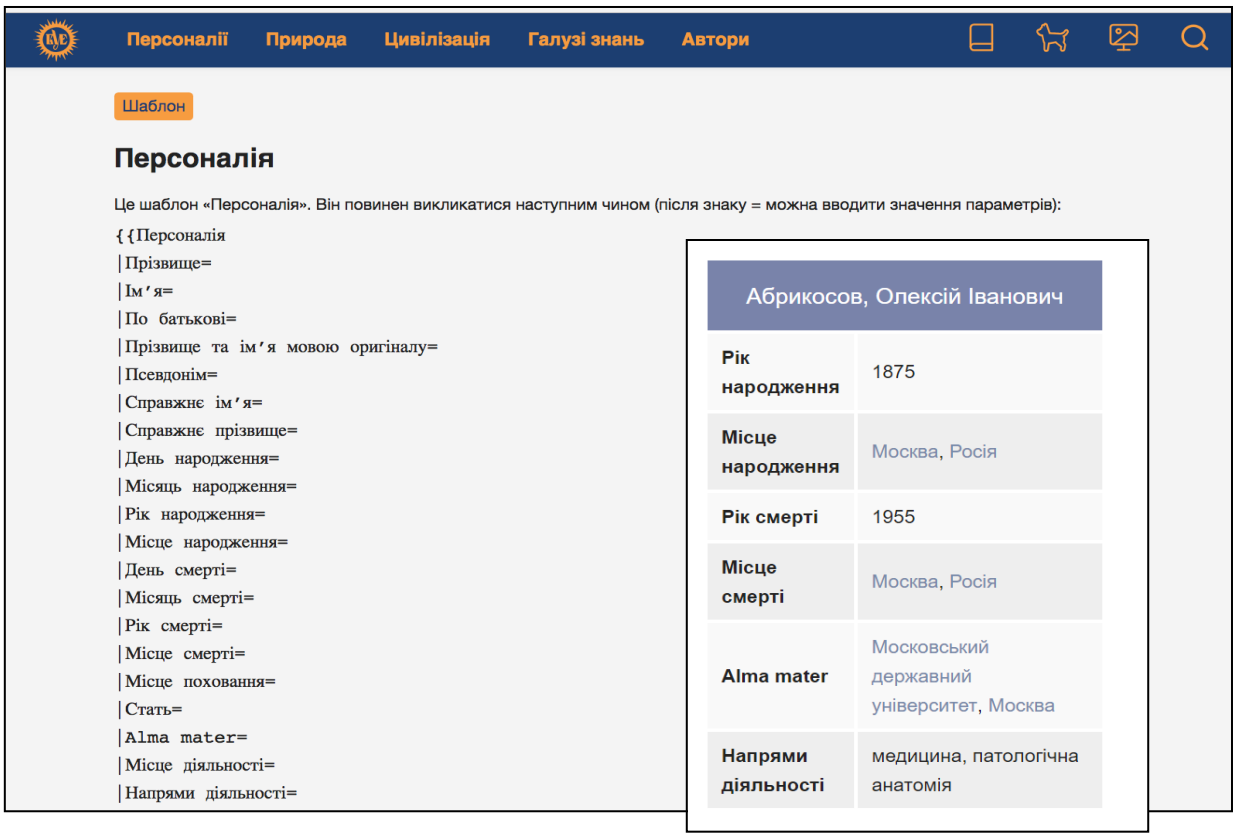


Рис. 3. Шаблон е-ВУЕ “Персоналія” та приклад його використання

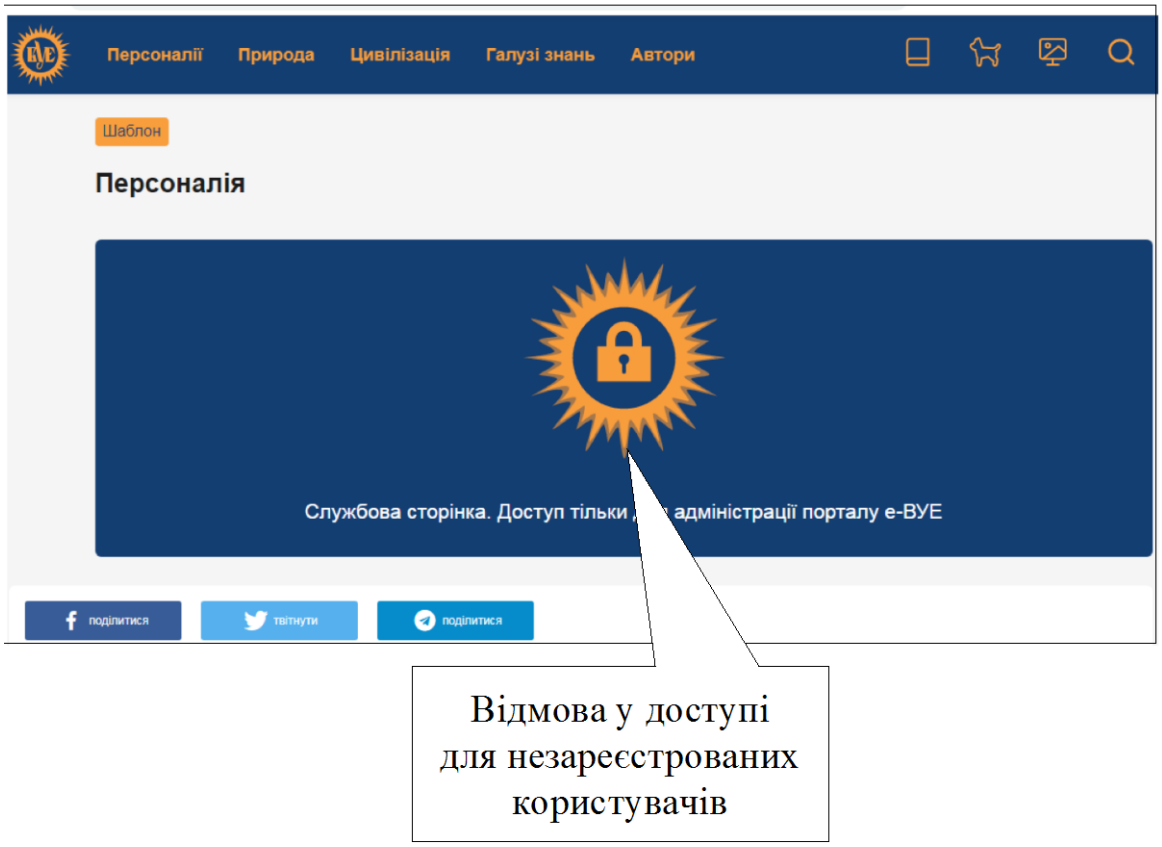


Рис. 4. Відмова у доступі до контенту сторінки шаблон е-ВУЕ “Персоналія”

Висновки

Розглянуто перспективи та область застосування Wiki-технологій, проаналізовано структуру подання контенту та програмне забезпечення MediaWiki. Запропоновано абстрактну модель архітектури MediaWiki, яка формалізує відношення між основними складовими цього технологічного середовища. Визначено проблему, що пов'язана з необхідністю керування доступом до контенту Wiki-ресурсів відповідно до специфіки інформації, що міститься у таких ресурсах. Результати аналізу показали, що базові засоби MediaWiki не дозволяють отримати задовільне рішення цієї задачі. Тому виникає потреба у створенні спеціалізованого програмного забезпечення, яке базується на класифікації контенту з використанням окремих просторів імен, категорій, шаблонів та семантичних властивостей. які здобуваються з MediaWiki, та є незалежним від ядра MediaWiki, а базується на програмуванні аналізу контенту на рівні скіну.

Наведене рішення апробовано у розробці порталу е-ВУЕ [14] і використовує знання щодо структури бази знань цього порталу.

Література

1. MediaWiki. https://www.mediawiki.org/wiki/MediaWiki
2. Гришанова І.Ю., Рогушина Ю.В. Адаптація технологічних засад semantic mediawiki до потреб онлайн-версії великої української енциклопедії ВУЕ. *Традиції та сучасні концепти енциклопедичної справи в Україні*: колективна монографія / За ред. Киридон А.М. К.: Державна наукова установа «Енциклопедичне видавництво», 2018. С. 240–253. https://ev.vue.gov.ua/wp-content/uploads/ 2019/11/Traditions.pdf
3. Leuf B., Cunningham W. The Wiki way: collaboration and sharing on the Internet, 2001, http://www.citeulike.org/group/13847/article/7659081.
4. Wikipedia – https:// www. wikipedia.org.
5. Рогушина Ю.В., Прийма С.М., Строкань О.В. Створення та використання семантичних Wiki-ресурсів: навчальний довідник. Мелітополь, ФОП Однорог Т.В., 2017. 169 с.
6. Manual: What is MediaWiki? https://www.mediawiki.org/wiki/Manual:What_is_MediaWiki%3F.
7. Brown A., Wilson G. The Architecture of Open Source Applications: Elegance, Evolution, and a Few Fearless Hacks, V. 1, 2011. ftp://188.235.129.151/incoming/books/Wilson%20G.%20-%20The%20Architecture%20Of%20Open%20Source%20Applications%20-%202011.pdf.
8. Manual: Skins. https://www.mediawiki.org/wiki/Manual:Skins.
9. Magnus Manske. https://en.wikipedia.org/wiki/Magnus_Manske
10. Авторизація. https://uk.wikipedia.org/wiki/Авторизація
11. Документація для розробників скінів https://www.mediawiki.org/wiki/Manual:Skinning_Part_2
12. Рогушина Ю.В., Гришанова І.Ю. Онтологічна модель бази знань онлайн-версії «Великої української енциклопедії» та методи її застосування для семантичного пошуку та навігації. Енциклопедичний контент і виклики сучасного світу: Збірник матеріалів наукової конференції / За ред. Киридон А.М. К.: Державна наукова установа «Енциклопедичне видавництво», 2019. С. 69–74.
13. Гладун А.Я., Рогушина Ю.В. Репозитории онтологий как средство повторного использования знаний для распознавания информационных объектов. *Онтология проектирования.* 2013. № 1 (7). С. 35–50.
14. Велика українська енциклопедія. https://vue.gov.ua/

References

1. MediaWiki. https://www.mediawiki.org/wiki/MediaWiki
2. Grishanova I.Y, Rogushina J.V. (2018) Adaptation of technological means of Semantic Mediawiki for needs of online version of Great Ukrainian Encyclopedia // Encyclopaedias in Ukraine: people, ideas, steps: collective monograph / Ed. Kyrydon A.M., Kyiv, P.240-253. [in Ukrainian]
3. Leuf B.,Cunningham W. (2001) The Wiki way: collaboration and sharing on the Internet, 2001, http://www.citeulike.org/group/13847/article/7659081.
4. Wikipedia. https:// www. wikipedia.org.
5. Rogushina Y.V., Priyma S.M, Strokan O.V. (2017) Creating and use of the Semantic Wiki

resources: tutorial. - Melitopol, FOP Odinorog T.V. – 169 p. [in Ukrainian]
6. Manual: What is MediaWiki? https://www.mediawiki.org/wiki/Manual:What_is_MediaWiki%3F.
7. Brown A., Wilson G. (2011). The Architecture of Open Source Applications: Elegance, Evolution, and a Few Fearless Hacks (V. 1). – ftp://188.235.129.151/incoming/books/Wilson%20G.%20-%20The%20Architecture%20Of%20Open%20Source%20Applications%20-%202011.pdf.
8. Manual: Skins. https://www.mediawiki.org/wiki/Manual:Skins.
9. Magnus Manske. https://en.wikipedia.org/wiki/Magnus_Manske.
10. Avtorization. https://uk.wikipedia.org/wiki/Авторизація.
11. Documentation for skin developers. – https://www.mediawiki.org/wiki/Manual:Skinning_Part_2.
12. Grishanova I.Y, Rogushina J.V. (2019) Ontological model of online version of Great Ukrainian Encyclopedia knowledge base and methods of its use for semantic search // Encyclopaedic content and challenges of modern world / Ed. Kyrydon A.M., Kyiv. P. 64–74. [in Ukrainian]
13. Gladun A., Rogushina J. Ontology repositories as a means of knowledge reusing for recognition of information objects // Ontology for Design. 2013. N 1 (7). P. 35–50. [in Russian]
14. Great Ukrainian Encyclopedia, https://vue.gov.ua/

Одержано 05.02.2020

Про авторів:

Гришанова Ірина Юріївна,
науковий співробітник.
Кількість наукових публікацій в українських виданнях – 19.
Кількість наукових публікацій в зарубіжних виданнях – 3.
http://orcid.org/0000-0003-4999-6294,

Рогушина Юлія Віталіївна,
кандидат фізико-математичних наук,
старший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 155.
Кількість наукових публікацій в зарубіжних виданнях – 31.
http://orcid.org/0000-0001-7958-2557.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03181, Київ-187,
проспект Академіка Глушкова, 40.
Тел.: 066 550 1999.

E-mail: ladamandraka2010@gmail.com,
i26031966@gmail.com

УДК 004.05, 004.4[2+9], 004.94, 519.7 https://doi.org/10.15407/pp2020.01.047

В.М. Яковлев

АЛГЕБРАЇЧНІ ШАБЛОНИ ВРАЗЛИВОСТЕЙ БІНАРНОГО КОДУ

Пошук вразливостей у програмному забезпеченні є на поточний час актуальним завданням та джерелом наукових викликів. Описаний у статті алгебраїчний підхід покликаний збільшити ефективність та достовірність алгоритмів пошуку. Запропоновано засоби формального опису поведінки бінарного коду та вразливостей в термінах алгебри поведінок, а методику створення шаблонів вразливостей бінарного коду.

Ключові слова: вразливості програмного забезпечення, символьне моделювання, алгебраїчне зіставлення, алгебра поведінок.

Вступ

Однією з найважливіших сучасних проблем в галузі інформаційних технологій є виявлення вразливостей в бінарному коді. Проблема настільки серйозна, що дослідження в цьому напрямку організовуються й фінансуються на рівні державних структур. Зокрема, в США в 2018 році на базі Національного Директорату з Захисту та Програмах (National Protection and Programs Directorate, NPPD) створено Агентство з Кібербезпеки та Інфраструктури (Cybersecurity and Infrastructure Security Agency, CISA), наділене широкими правами щодо захисту програмних систем та комп'ютерних мереж. У 2016 році створено Європейську Організацію з Кібербезпеки (European Cyber Security Organisation, ECSO), а в квітні 2019 року Рада Європи схвалила Акт з Кібербезпеки (Cybersecurity Act), який, зокрема, встановлює загальноєвропейські правила сертифікації комп'ютерних систем та створює загальноєвропейське Агентство з Кібербезпеки на базі існуючого Загальноєвропейського Агентства з Мережевої та Інформаційної безпеки (European Union Agency for Network and Information Security, ENISA).

Американське агентство DARPA в 2016 році започаткувало спеціальний конкурс, Cyber Grand Challenge [1], що заохочує дослідницькі організації до створення систем для автоматизованого, масштабованого та швидкісного програмного забезпечення для виявлення вразливостей та кіберінфекцій. Всі троє переможців конкурсу 2016 року в своїх продуктах застосовували алгебраїчний підхід, яких дозволяє більш ефективно реалізувати алгоритми виявлення вразливостей.

Переможець конкурсу Mayhem [2], розроблений командою ForAllSecure з Пітсбургу, включає інструменти, які використовують символічне виконання, зокрема, комбінований (concolic) підхід, що підвищує ефективність обходу програмних шляхів. Крім того, індексована пам'ять та увід користувача розглядаються як символьні значення, що забезпечило більш повне покриття програмного середовища.

Команда Xandra TECHx [3] з Ітаки, штат Нью-Йорк, та Шарлоттсвіля посіла друге місце у загальному заліку в фінальній події. Тим не менш, ця команда була першою у вирішенні проблеми виявлення та запобігання визискам (exploit) за рахунок використання символьного інструменту пошуку.

Система Mechanical Phish [4], розроблена командою «Шелфи» з Санта-Барбари, Каліфорнія, також використовує символьне виконання з розмиванням вхідних даних.

Також існує багато інших подібних інструментів, які працюють з бінарним кодом, використовують символьні техніки моделювання. До них належать S2E [5] та інші системи з обмеженим символьним виконанням, такі як CUTE [6] та Клі [7].

В даній статті пропонується застосування алгебраїчного підходу до виявлення вразливостей за допомогою алгебраїчних моделей та виявлення поведінок, які

б відповідали певним зразкам у бінарному коді. Ця методика базується на алгебраїчній системі програмування (APS), і покладена в основу прототипу алгебраїчної системи виявлення вразливих місць у бінарному коді з використанням алгебраїчного методу відповідності.

1. Алгебра поведінок

Вразливість розглядається як поведінка системи, яка потенційно дозволяє зловмиснику виконувати такі дії, як, наприклад, пошкодження або крадіжка даних. Для конкретизації такої поведінки використовується певний алгебраїчний формалізм. Прикладом такого формалізму є алгебра поведінок, представлена Девідом Гілбертом та Олександром Летичевським [8].

Розглянемо множини поведінок та дій. Кожна поведінка складається з певних дій та іншої поведінки. Алгебра поведінок – це двосортна алгебра над множинами поведінок та дії з наступними операціями:

- операція префіксингу `a.B` означає, що перед поведінкою `B` виконується дія `a`;
- операція недетермінованого вибору поведінок `u + v` встановлює альтернативну поведінку.

Алгебра включає три термінальні константи: успішне припинення Δ, тупик 0 та невідома поведінка ⊥. Алгебра поведінки також збагачена двома операціями: паралельні (||) та послідовні (;) композиції поведінок. Терм, створений з операцій алгебри поведінки над поведінками та діями, формує вираз алгебри поведінок. Будь-яка поведінка може бути представлена як набір рівнянь, які містять ім'я поведінки в лівій частині та вираз алгебри поведінок у правій. Наприклад, наведена далі поведінка визначає поведінку програми, яка друкує щось у циклі.

```
Program = initCycle.Cycle; Δ
Cycle = print.Cycle + endCycle
```

`initCycle`, `endCycle` та `print` – це дії, а `Program` і `Cycle` – поведінки.

Всі дії об'єкта, для якого описується поведінка, передбачає зміну його стану, що визначається як множина атрибутів, представлена у вигляді типізованих змінних або функцій. Кожна дія також визначається парою формальних виразів – передумовою та постумовою дії. Семантика цих виразів визначається складністю об'єкта, що формалізується. Наприклад, наведена далі дія передбачає можливість зміни стану об'єкта, визначеного атрибутами `A` і `B`:

```
Action(A, B) =
      (A> B) && !(A == 0) ->
            B = (B + 1) / A.
```

Семантика дії, яка представлена в С-подібному синтаксисі, означає, що якщо передумова `(A > B) && !(A == 0)` справедлива для конкретного значення `A` і `B` або підходить для символічних (довільних) значень з `A` і `B`, тоді ми можемо змінити атрибут `B`:

```
B = (B + 1) / A.
```

Дію можна параметризувати за атрибутами, що згадуються в її передумові дії. Для множини поведінок ми можемо визначити іншу поведінку високого рівня, яка міститиме усі інші поведінки.

Підстановка високорівневої поведінки для дій дає набір послідовностей дій, тобто різні сценарії зазначеної поведінки високого рівня. Ця процедура називається розгортанням поведінки. Набір формул над атрибутами визначає об'єкт на кожному кроці розгортання. Формули містять символьні атрибути, а процес розгортання називається символьним моделюванням.

2. Семантика бінарного коду

Розглянемо виконавчий модуль, який складається з набору інструкцій процесора, представлених у вигляді бінарного коду. Дизасемблювавши бінарний код, отримаємо набір асемблерний текст. У подальшому розглядатимемо асемблер Intel x86 [9]. Поведінка програми визначається послідовністю інструкцій, а різні сценарії виконання генеруються за рахунок зміни вхідних даних. У формальній моделі кожна інструкція відповідає певній дії.

Потік управління програмою може бути представлений виразами алгебри по-

ведінки. У поведінковому виразі `A = a.B`, `a` – позначає певну дію, яка відповідає поточній інструкції, а `B` – представляє поведінку решти програми, що виникає після дії `a`. В інструкціях, що містять альтернативи в контрольному потоці, потрібна операція "+". Приклад поведінкових виразів і відповідний код показані далі:

```
B8049865 = sub(1,eax,0x81d01e0).B804986a,
B804986a = sar(1,eax,0x2).B804986d,
B804986d = mov(1,edx,eax,mov).B8049876,
B8049876 = jmp(1,jne).B8049879
```

```
8049865:  2d e0 01 1d 08          sub eax,0x81d01e0
804986a:  c1 f8 02                sar eax,0x2
804986d:  89 c2                   mov edx,eax
8049876:  75 01                   jne 8049879
```

Параметризовані дії та імена поведінок позначаються відповідно інструкціям та адресам операторів в асемблерному лістингу. Кожна дія моделі змінює її стан, який визначається певною формулою в атрибутному середовищі. Розглянемо структуру такого середовища.

З урахуванням структури новітніх процесорів Intel, середовище складається з наступних компонентів:

- набору регістрів загального та спеціального призначення. Деякі атрибути ідентифікуються за іменами регістрів: `ax, al, bx, bl, .., eax, ebx,…, rax, rbx,…, ebp, esp, rbp, rsp, rip`;
- фізичної пам'яті, яку можна розглядати як функцію `Memory(addr)`, де `addr` визначає адресу наявної пам'яті.

Семантика інструкції визначається передумовою і відповідною зміною середовища. Наприклад, якщо розглянути семантику інструкції `cjne A B z`, то передумова визначається значеннями операндів інструкції, а поведінка описується диз'юнкцією:

```
Bx1 = cjne.Bz +!cjne.Bx2
Bx2 =…
```

Дії:

```
cjne(n,A,B) = !(A == B) ->
    PI = PI+z+3; FLAG_C = (B > A)
!cjne(n,A,B) = (A == B) ->
    PI = PI + 3;
```

Тобто, якщо два операнди рівні, переходимо до поведінки Bz, змінюємо вказівник на значення z+3 та признаємо булеве значення атрибуту FLAG_C. В іншому випадку змінюємо вказівник на 3 та переходимо до наступної інструкції. Після відповідної трансляції набори поведінок та дій можуть бути отримані з асемблерного лістинга автоматично.

3. Алгебраїчні шаблони вразливостей

Шаблон вразливості визначає поведінку програми, що призводить до стану вразливості. Треба розрізняти стан помилки програми та стан вразливості, оскільки останній, як правило, містить також дії, що відповідають введенню в програму певних даних, обробка яких призводить до «спрацювання» вразливості. Шаблон вразливості складається з виразів алгебри поведінки та відповідних дій. Загальна форма моделі вразливості – це поведінка:

```
VulnerabilityPattern = Input;
                       ProgramBehavior;
                       VulnerabilityPoint
```

Наведений шаблон складається з трьох узагальнених поведінок.

`Input` – це поведінка програми, яка представляє інструкції, що діють від зовнішнього середовища. Вона може включати введення даних з командного рядку, зчитування файлів, отримання даних з мережі тощо. Переважно, введення реалізується як системний виклик основного коду операційної системи.

`ProgramBehavior` – це частина програми, яка виконується між точками введення даних та вразливості. В описі поведінки може бути більше однієї такої поведінки.

`VulnerabilityPoint` представляє дії, які тягнуть за собою «спрацювання» вразливості. Узагальнена поведінка охоплює всі можливі сценарії або послідовності інструкцій, що призводять до вразливості. Це є найскладніша частина формалізації

структури вразливості з огляду на необхідність зібрати всі можливі сценарії. Наприклад, копіювання пам'яті може бути представлено декількома способами та, відповідно, різними послідовностями інструкцій. Одним із таких способів є копіювання областей пам'яті в циклі.

```
writeMemory =
    mov(1,
        GeneralRegister,
        MemoryOperand).
    mov(2,
        MemoryOperand,
        GeneralRegister).
    add(1,GeneralRegister,1).
    add(2,GeneralRegister,1).
    X3;
    (writeMemory + Delta)
```

Така поведінка означає, що запис у пам'ять виконується в циклі, і кінець поведінки відповідає його завершенню. Параметр `GeneralRegister` посилається будь-який регістр загального призначення, а `MemoryOperand` позначає вміст фізичної пам'яті. `X3` є довільною поведінкою.

Відповідні дії виглядають наступним чином:

```
mov(1,GeneralRegister,
      MemoryOperand) = 1 ->
         GenPurpose(x) &&
         x = Memory(y),
mov(2 MemoryOperand,
      GeneralRegister) = 1 ->
         Memory(z) = x,
add(1, r, 1) = 1-> y = y + 1
add(2, r, 1) = 1-> z = z + 1
```

Цей простий приклад поведінки має на увазі, що вміст фізичної пам'яті за адресою y передається в регістр x і предикат `GenPurpose (x)` дорівнює `true` або `x == ax || x == bx || x == cx || x == dx` для дії `mov (1,…)`. Дія `mov (2,…)` передає вміст регістру `x` у фізичну пам'ять за адресою `z`. У діях `add (1,…)` та `add (2,…)` ми збільшуємо адреси, за якими ми звертаємося до пам'яті. Змінна `r` означає, що не має значення, де саме знаходяться адреси, за якими ми звертаємося до областей пам'яті – в регістрі загального призначення або в пам'яті. При розгортанні поведінки враховуються лише необхідні нам об'єкти, зазначені у виразах алгебри поведінки.

Розглянемо декілька узагальнених шаблонів вразливостей, пов'язаних з переповненням буфера пам'яті, що є найбільш розповсюдженою причиною вразливостей програм.

Поведінка `Input`, що відповідає введенню даних у програму, в загальній формі виглядає як

```
Input=mov(i,
          GeneralRegister,
          MemoryAddr).
      X1.
      mov(j,
          Memory(x),
          GeneralRegister).
      Input,
mov(i,
    GeneralRegister,
    MemoryAddr)=
      1-> Dirty(GeneralRegister),
mov(j,Memory(x),GeneralRegister)=
      Dirty(GeneralRegister) ->
          Dirty(x) &
          !Dirty(GeneralRegister)
```

В сучасних операційних системах введення даних у програмний буфер здійснюється шляхом копіювання в цей буфер даних з системного буфера, представленого прямою адресою `MemoryAddr`. Оскільки в одноадресній архітектурі Intel будь-яке копіювання має відбуватися через регістр загального призначення, маємо дві дії, перша з яких копіює байт з системного буфера в регістр, а друга – з регістра в локальний програмний буфер за адресою `x`.

При цьому ми відмічаємо вміст регістра, і далі вміст локального буфера як `Dirty`.

Треба зазначити, що наведена поведінка `Input` відповідає найпростішому варіанту реалізації введення даних, оскільки в розширеній множині інструкцій Intel існують інструкції копіювання пам'ять-пам'ять, наприклад, `movs`. Таким чином, поведінка `Input` має бути доповнена ін-

шими існуючими реалізаціями алгоритму введення даних.

Тепер розглянемо розповсюджений сценарій вразливості, пов'язаний з переповненням буфера, розташованого в області стека. «Спрацювання» такої вразливості може призвести до порушення роботи програми, або, при певних умовах, до виконання зловмисного коду.

```
StackVulnerability =
    (mov(l,
         Memory(s),
         GeneralRegister).
     Delta +
     !mov(l,
         Memory(s),
         GeneralRegister)).
    StackVulnerability,
```

Вказана поведінка означає, що з регістрів пам'ять у циклі пишеться у пам'ять, яка може вміщувати експлойт.

```
mov(l, Memory(s),
    GeneralRegister) =
       Dirty(GeneralRegister) &
      (s <= BP)     -> 1,
mov(m, Memory(s),
    GeneralRegister) =
       !(m = l) &
       Dirty(GeneralRegister)
                  -> Dirty(s),
mov(n, Memory(s),
    GeneralRegister) =
       !(n = l)  & (Dirty(s) &
       !Dirty(GeneralRegister))
                  -> !Dirty(s)
mov(m, GeneralRegister,
    Memory(s)) =
       !(m = l) & Dirty(s) ->
          Dirty(GeneralRegister),
mov(n, GeneralRegister,
    Memory(s))=
       !(n = l) & (!Dirty(s) &
       Dirty(GeneralRegister)) ->
          !Dirty(GeneralRegister)
```

Наведена поведінка відповідає одній з реалізацій процедури копіювання буфера вводу в буфер, розташований у стеку. «Спрацювання» вразливості відбувається, коли для дії `mov(l, Memory(s), GeneralRegister)` спрацьовує передумова `Dirty(GeneralRegister) & (s<=BP)`, що є переповненням стеку.

Інший вірогідний сценарій вразливості пов'язаний із зверненням до пам'яті, розташованої поза межами буфера, розташованого в області динамічної пам'яті, або «купи».

```
HeapVulnerability = AllocHeap;V2;
                    AccessHeap,
```

Дві важливі частини поведінки описують виділення пам'яті в «купі» та звернення до пам'яті в «купі» з метою читання або запису.

Шаблон, що відповідає поведінці виділення пам'яті, виглядає наступним чином:

```
AllocHeap = mov(I,AX,y),
            V3;
            mov(j,AX,z),
mov(I,AX,y)=1->N=AX,
mov(j,AX,z) = 1->addr=AX;
          Forall( a+N > j >= a,
                  Allocated(j) ),
```

Наведений шаблон описує операцію виділення `N` байт пам'яті. Функція виділення пам'яті приймає значення `N` як аргумент, передаючи його через регістр `AX`. Після звернення до менеджера пам'яті адреса виділеної області повертається через згаданий регістр. Одночасно, модель маркує виділені комірки пам'яті як `Allocated`. Нарешті,

```
AccessHeap=ReadHeap+WriteHeap,
ReadHeap=(mov(k,
              GeneralRegister,
              h).
          Delta +
          !mov(k,
               GeneralRegister,
               h)).
          ReadHeap,
mov(k, GeneralRegister, h) =
   Allocated(h) & (h >=N | h < 0)
                              -> 1
WriteHeap=(mov(l,
               Memory(h),
               GeneralRegister).
           Delta +
           !mov(l,
```

```
                Memory(h),
                GeneralRegister)
          ).WriteHeap,
mov(l,
    Memory(h),
    GeneralRegister) =
         Allocated(h) &
         (h >= N | h < 0) -> 1,
```

Таким чином, наведені поведінки «спрацьовують», коли операції читання або запису звертаються до пам'яті поза межами виділеного блоку.

Ще один варіант – поведінка, що описує звернення до статичної пам'яті (зовнішні змінні), дещо відрізняється від наведеної для динамічної пам'яті. Нагадаємо, такі блоки пам'яті існують в програмі впродовж всього часу її роботи.

```
AccessStatic=ReadStatic+
             WriteStatic,
ReadStatic=lea(x,
               GeneralRegister,
               offset).
           V4;DoRead,
DoRead=(mov(y,GeneralRegister,s).
        Delta +
        !mov(y,GeneralRegister,s)
       ).DoRead,
WriteStatic=lea(x,
                GeneralRegister,
                offset).
            V5;
            mov(y,
                GeneralRegister,
                s),
lea(x,GeneralRegister,offset)=
               1->Size(s)=offset,
mov(y,GeneralRegister,s)=
         s>=Size(s) | s < 0 -> 1,
```

Як правило, передумовою звернення до блоку статичної пам'яті є налаштування певного регістру на адресу цього блоку, що здійснюється через інструкцію `lea(…)`, а подальші дії аналогічні тим, що відбувається при зверненні до пам'яті інших типів.

Очевидно, наведені приклади шаблонів вразливостей описують дані вразливості лише частково, тому що існують інші сценарії подібних дій. Більш того, існує багато інших видів вразливостей, і створення відповідних шаблонів є предметом подальших досліджень.

4. Алгебраїчне зіставлення

Процедура алгебраїчного зіставлення до певної міри схожа на традиційний пошук відповідностей, що використовується в антивірусних програмах. Традиційні шаблони програмного коду містять конкретні значення, які співставляються з кодом, що перевіряється, тоді як алгебраїчне зіставлення передбачає вирішення або доведення кожного кроку зіставлення. Ця властивість алгебраїчного підходу дозволяє охопити більше випадків вразливостей та значно підвищує точність виявлення, зменшуючи кількість випадків хибного виявлення.

Алгебраїчне зіставлення виконується під час символьного моделювання бінарного коду. При зіставленні на рівні поведінок на кожному кроці виявляється істинність виразу `Env && Prec`, де `Env` – символьне середовище, а `Prec-` передумова відповідної дії. У такому випадку виконується формула постумови дії у середовищі шаблона та в середовищі моделі програми. Символьне моделювання виконується до виявлення вразливостей або вичерпання шляхів у просторі пошуку.

Висновки

Основна проблема алгебраїчного підходу полягає у тому, що досяжність у загальному випадку не визначається. Типовими проблемами також є експоненційний вибух простору станів або можливих програмних сценаріїв. Такі проблеми можуть вирішуватися за допомогою альтернативних символьних методів, таких як генерація інваріантів, апроксимація або зворотне символьне моделювання. За допомогою налаштувань пошуку можливо зменшити простір станів, наприклад, визначити певне обмежене покриття рядків коду. З іншого боку, таке звуження простору пошуку може спричинити до того, що деякі вразливості буде пропущено.

Іншою проблемою є узагальнення алгебраїчних шаблонів. Власне узагаль-

нення засновується на аналізі великої кількості прикладів та визначенні загальних рис можливих сценаріїв. Оскільки наразі універсальна методика побудови та узагальнення шаблонів відсутня, немає гарантій того, що побудований шаблон охоплює всі можливі сценарії.

Тим не менш, важлива перевага алгебраїчного підходу полягає у тому, що вразливості можуть бути виявлені набагато точніше порівняно з традиційними підходами. Крім того, описи вразливостей можуть охоплювати цілі сімейства можливих сценаріїв виконання коду.

Ефективність виявлення вразливостей може бути підвищена за рахунок реалізації зіставлення на декількох рівнях, наприклад, на рівні тільки поведінок, що може виконуватися досить швидко, і лише по тому – на рівні моделювання поведінок з доведенням досяжності на сильно звуженому просторі пошуку.

З огляду на проведені численні експерименти з робочим прототипом системи алгебраїчного зіставлення на основі алгебраїчної системи APS [10] згаданий підхід довів свою перспективність, що обумовлює необхідність подальших досліджень.

Література

1. DARPA, "Cyber Grand Challenge." [Online]. Available: https://www.cybergrandchallenge.com/.
2. Cha S.K., Avgerinos T., Rebert A., Brumley D., "Unleashing Mayhem on binary code," Proceedings. IEEE Symposium on Security and Privacy. 2012. P. 380–394.
3. Nguyen-Tuong A., Melski D., Davidson J.W., Co M., Hawkins W., Hiser J.D., Morris D., Nguen D., and Rizzi E. "Xandra: An autonomous cyber battle system for the Cyber Grand Challenge," IEEE Security & Privacy. 2008. Vol. 16. N. 2. P. 42–53.
4. Mechaphish, "Github repository." [Online]. Available from:https://github. com/ mechaphish/mecha-docs.
5. Chipounov V., Kuznetsov V., Candea G. "S2E: A platform for invivo multi-path analysis of software systems." Asplos. 2011. Vol. 46. P. 1–14.
6. Sen K., Marinov D., Agha G., Sen K., Marinov D., Agha G. "CUTE: A concolic unit testing engine for C." 10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE'05). 2005. Vol. 30. N 5. P. 263.
7. Cadar C., Dunbar D., Engler D.R. "KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs," Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. 2008. P. 209–224.
8. Gilbert D., Letichevsky A. "Interaction of agents and environments," Recent trends in algebraic development technique, LNCS 1827 (D. Bert and C. Choppy, eds.), Springer-Verlag, 1999.
9. Intel 64 and IA-32. "Architectures software developer's manual." Intel Corporation. 1997–2016.
10. Algebraic Programming System, APS, [Online]. www.apsystem.org.ua

References

1. DARPA, "Cyber Grand Challenge." [Online]. Available: https://www.cybergrandchallenge.com/.
2. Cha S.K., Avgerinos T., Rebert A., Brumley D., "Unleashing Mayhem on binary code," Proceedings. IEEE Symposium on Security and Privacy. 2012. P. 380–394.
3. Nguyen-Tuong A., Melski D., Davidson J.W., Co M., Hawkins W., Hiser J.D., Morris D., Nguen D., and Rizzi E. "Xandra: An autonomous cyber battle system for the Cyber Grand Challenge," IEEE Security & Privacy. 2008. Vol. 16. N. 2. P. 42–53.
4. Mechaphish, "Github repository." [Online].
5. Available from:https://github. com/ mechaphish/mecha-docs.
6. Chipounov V., Kuznetsov V., Candea G. "S2E: A platform for invivo multi-path analysis of software systems." Asplos. 2011. Vol. 46. P. 1–14.
7. Sen K., Marinov D., Agha G., Sen K., Marinov D., Agha G. "CUTE: A concolic unit testing engine for C." 10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on

Foundations of Software Engineering (ESEC/FSE'05). 2005. Vol. 30. N. 5. P. 263.
8. Cadar C., Dunbar D., Engler D.R. "KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs," Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. 2008. P. 209–224.
9. Gilbert D., Letichevsky A. "Interaction of agents and environments," Recent trends in algebraic development technique, LNCS 1827 (D. Bert and C. Choppy, eds.), Springer-Verlag, 1999.
10. Intel 64 and IA-32. "Architectures software developer's manual." Intel Corporation. 1997–2016.
11. Algebraic Programming System, APS, [Online]. www.apsystem.org.ua

Одержано 30.01.2020

Про автора:

Яковлев Віктор Михайлович,
провідний математик.
Кількість наукових публікацій в українських виданнях – 7.
Кількість наукових публікацій в зарубіжних виданнях – 1.
http://orcid.org/ 0000-0001-6000-5215.

Місце роботи автора:

Інститут кібернетики імені В.М. Глушкова НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.

E-mail: victoryakovlev@ukr.net

УДК 004.05 https://doi.org/10.15407/pp2020.01.055

О.О. Летичевський, С.О. Горбатюк

ДЕЦЕНТРАЛІЗОВАНІ СИСТЕМИ В ЛОГІСТИЦІ: ОГЛЯД ВИКОРИСТАННЯ ТА ПРОБЛЕМИ БЕЗПЕКИ

Проаналізовано напрямки застосування та використання децентралізованих (розподілених) систем на прикладі технології блокчейн в міжнародній логістиці. Розглянуто перспективи та ключові аспекти подальшого розвитку децентралізованих систем та великих баз даних. Розглянуто основи функціонування смарт-контрактів на базі Etherium та мови Solidity. Проаналізовано основні проблеми безпеки в цілісному ланцюгу постачання, виявлено напрямки покращення відслідковування безперервності контролю якості товару під час руху в мультимодальному ланцюгу постачання. Запропоновано методи посилення контролю за безпекою в децентралізованих системах у логістиці. Розглянуто методологію формальних алгебраїчних методів з метою аналізу та дослідження властивостей перевезень при взаємодії агентів певного логістичного середовища.

Ключові слова: децентралізована система, блокчейн, смарт-контракт, логістика, Solidity.

1. Загальна інформація

У зв'язку із зростаючими обсягами промислового виробництва, ростом населення, глобалізацією та міжнародною торгівлею, у світі зростає роль міжнародної логістики та все більше уваги приділяється до зростання її ефективності та вирішення проблем.

IT сектор також не міг лишитись у стороні від глобальних трендів, тенденцій та широко залучається до міжнародної логістики. Одна з таких прогресивних IT технологій, що набула популярності протягом останнього десятиліття – це технологія блокчейн [1, 2], що характеризується принципом децентралізації. Блокчейн технологія є технологією загального, надзвичайно широкого призначення [3] та може застосовуватись в будь-якій галузі промисловості. Згідно визначенню автора цієї технології Сатоші Накамото, це «розподілена база даних з записами або публічний реєстр всіх транзакцій або цифрових подій, що були виконані та поширені серед всіх учасників» [1]. Система надійна та захищена – записи перевіряються публічними реєстрами та ніколи не можуть бути видаленими в майбутньому, тобто зберігаються завжди. Головні переваги технології блокчейн – це прозорість та висока ефективність [4].

Нині контроль за товарооборотом та документообігом здійснюється через багатьох посередників, які збирають значну комісію за свою діяльність, а також витрачають деякий час та ресурси на здійснення цієї діяльності. Технологія блокчейн дає змогу всім зацікавленим сторонам робити транзакції та здійснювати контроль безпосередньо один між одним, при чому це здійснюється дуже швидко та безпечно. Іншою перевагою технології блокчейн є можливість виконувати «смарт контракти» («smart contracts»). «Смарт контракти» – комп'ютерні програми, які створені для автоматичного виконання контрактів чи обміну інформацією [1].

В мультимодальних перевезеннях у міжнародній логістиці використання технології блокчейн у розрізі швидкості та надійності контролю, та обміну інформацією виглядає особливо перспективно, так як там задіяно безліч учасників з різними функціями в різних країнах світу (виробник товару, склад, експортер, митниця та контролюючі органи держави, власники засобів перевезень, портові оператори, оператори перевантаження, імпортери, продавці та покупці, проміжні власники товарів). Тому оперативний контроль та реагування має особливе значення.

Технологія блокчейн викликала інтерес та швидко розвивається в багатьох галузях: енергетика, охорона здоров'я, фінанси, державний контроль та інше. Не виключенням є і міжнародна логістика, проте, на жаль вивчення можливостей застосування технології блокчейн в міжнародній логістиці не є достатнім [5]. Проте,

без сумнівів, міжнародна логістика – одна із галузей, в якій технологія блокчейн має надзвичайні перспективи застосування, тому кількість наукових досліджень буде зростати.

Проте ряд скандинавських країн вже виявило зацікавлення та використовують блокчейн у своїй діяльності. Це пов'язано з тим, що вони в силу обставин вимушені використовувати в основному міжнародні морські перевезення. Ряд значних міжнародних морських операторів вже почали використовувати технологію блокчейн та вивчають можливості більш широкого використання [6].

Також міжнародне судоходство – це галузь, яка жорстко регулюється Міжнародною Судоходною Організацією (International Maritime Organization – IMO) та законодавством Евросоюзу [7]. Законодавство стає щороку все суворішим, вимоги все вищими, тому ведучі морські перевізники виявили зацікавленість в подальшому розвитку технології блокчейн для оптимізації своїх бізнес-процесів.

Згідно з дослідженням міжнародної сертифікаційної та класифікаційної спільноти DNV GL один незапланований простій під завантаження-вивантаження чи запізнення може коштувати логістичному оператору $2–5 млн. в день. Приблизно 50 % трапляються через різноманітні механічні поломки. Але недостатній менеджмент інформації призводить до прихованих втрат, який може вимірюватись у розмірі до 20 % операційного бюджету [8]. Тому вони заявляють про своє бажання вивчати та розробляти технологію великих масивів інформації у розподілених системах з подальшим її застосуванням в автоматизованій системі менеджменту ризиків використання активів. Вони називають свою технологію «cloud-based digital twin» (цифрові двійники в хмаровому сховищі).

На сьогоднішній день потік інформації типової системи постачання складає близько 100 гігабайт в день [9], за оцінками експертів цей потік інформації зросте до 35 зетабайт до 2020 [10].

Доктор Мартін Стопфорд, один з передових світових експертів та аналітиків в економіці та морських перевезеннях, заявляє, що «діджіталізація» (перехід в цифровий світ) є єдиним та невідворотним виходом для галузі морських перевезень. Аналізуючи економічні цикли, розвиток судобудівництва та перевезень з початку 20 століття, він заявляє, що три методи здатні змінити бізнес-модель циклів перевезень за допомогою діджіталізації, яка так необхідна галузі: «розумні» кораблі, «розумний» флот (з інтегрованою системою відслідковування та контролю торгового флоту як єдиної цілісної системи) та «розумна» глобальна логістика (з інтегрованою системою постачання «від-дверей-до-дверей»). Він додає, що великі культурні зміни повинні передувати таким змінам та для подальшого застосування додатків за технологією блокчейн у морській логістиці. Середнє комерційне судно має близько 2000 різних датчиків та сенсорів, але вони використовуються в малій мірі, в основному людьми, а повинні контролюватись автоматизованою системою. Така система передбачалась ще в 70–80 роки, але лише нині системи GPS-навігації та блокчейн технології дозволяють втілити їх в життя. Він заявляє, що «мати технологію – це перший крок, проте масиви інформації повинні використовуватись заради того, щоб показати зростання показників продуктивності та ефективності галузі» [11].

На сьогодні практично кожне вантажне судно, що здійснює міжнародні перевезення, оснащене системою ідентифікації за радіочастотами (Radio Frequency Identification – RFID). Ця система інтегрована з «інтернетом речей», початково для відслідковування місцезнаходження судна, а потім послужила для більш широкого застосування, та нині є основою для передових технологій на основі технології блокчейн [12]. Радіочастотне розпізнавання здійснюється за допомогою закріплених за об'єктом спеціальних міток, що несуть ідентифікаційну та іншу інформацію. Цей метод вже став основою побудови сучасних безконтактних інформаційних систем, і має стійку назву RFID-технології. Використання RFID-міток, за прогнозами, зросте до 209 мільярдів одиниць до 2021 року [13], а технології, що використовують такі мітки, дадуть змогу знизити операційні

витрати на 10–25 % [14]. Проте, найбільш вживаною технологією для Інтернету речей є GPS-навігація, що дозволяє з найбільшою точністю слідкувати за місцеперебуванням об'єктів [15].

У березні 2017 року Норвезький Університет Науки та Технологій разом з комерційними партнерами Kongsberg Seatex, Marintek, Maritime Robotics та Rolls-Royce Marine у водах фьйорду Трондхейм розпочали пілотний проект з запуску першого в світі автоматизованого самокерованого судна без жодної людини на борту, супроводжувати який будуть повітряні та підводні дрони.

Ще одним прикладом є портові оператори. Морські перевізники та інші логістичні оператори жорстко конкурують за місце в порту та портовому складі, створюючи при цьому величезні черги, але зібрана інформація не поширюється між пірсами навіть одного порту. Тому системі надзвичайно важко проаналізувати та розподілити інформацію від порту до зацікавлених сторін з метою оптимізації використання доступного ресурсу обслуговуючої логістики. Таким чином порти стають «вузьким місцем» логістичних операцій, а обсяг таких операцій щодня зростає. Порт Гамбург оцінює, що число обороту контейнерів, що проходять через порт, зросте до 25 мільйонів у рік до 2015 року, тому обробка такого величезного обсягу потребує залучення нових технологій обробки великих масивів інформації, швидкого та надійного розподілу інформації між зацікавленими сторонами, що, без сумніву, під силу блокчейн-системам [15].

Ще один інноваційний проект в міжнародній логістиці розпочав Морський та портовий департамент порту Сінгапур під назвою Smart Port Challenge. Вони співпрацюють з портом Ротердам, який розпочав подібну ініціативу (проект Onboard). Вони залучають Інтернет речей до індустрії морських перевезень, пропонуючи відкриту платформу з повністю інтегрованим програмним забезпеченням для спостереження в реальному часі за рухом суден та портовими операціями [16].

Більшість сучасних новітніх розробок фокусується на аналізі великих баз даних та на штучному інтелекті з метою зниження споживання палива та інших ресурсів, зниження викидів CO2 в атмосферу, оптимізації маршрутів, скорочення затрат часу та праці, інтеграцію з іншими популярними технологіями, такими як дрони та проектування вищезгаданих кораблів без людей на борту.

Проте бази даних, аналітичне програмне забезпечення, додатки, що базуються на Інтернеті речей потребують діяльності посередників у ланцюгу доступу, що додає лишніх кроків, затрат ресурсів та коштів. Помітно, що в галузі зростає увага до подальших інновацій та технологій. Разом з тим технологія блокчейн може бути тим рішенням, яке задовольнить зростаючі потреби міжнародної логістики, такі як децентралізація, кодування, безпечність, доступність, швидкодія.

2. Огляд використання блокчейн-рішень у логістиці

Велика фрахтова форвардна компанія Marine Transport International Limited (MTI) заявила, що з середини 2016 року використовує систему блокчейн відкритого типу під назвою TrustMeTM для контролю за відповідністю ваги брутто завантажених контейнерів, що надходять на завантаження, новим нормам постанов Міжнародної Морської Організації «SOLAS», які вступили в дію з липня 2016 року. Нове законодавство перекладає на відправника відповідальність за порушення норм гранично допустимої ваги брутто до етапу, коли контейнер поступає на завантаження на судно. Компанія почала використовувати технологію блокчейн TrustMeTM через необхідність надання бесперебійних та реальних актів перевірок портовим державним органам, перевізникам та власникам вантажу, таким чином уникаючи посередників при передачі інформації, приватних баз даних, логів та паперової роботи [5].

DNV GL – міжнародна сертифікаційна та класифікаційна спільнота, ключова компетентність якої є оцінка, консалтинг та менеджмент ризику в морській логістиці. DNV GL – найбільша класифікаційна спільнота, в реєстрі якої знахо-

диться 13175 суден та мобільних морських судоходних установок, сумарна водотонажність 265.4 млн. т., що являє собою 21 % світового ринку [17].

DNV GL помістила всі свої 90 000 сертифікатів у системи менеджменту та ланцюгів постачання в закриту систему блокчейн, став першою, хто це зробив в галузі сертифікації морської логістики. Кожен сертифікат має цифрову ідентифікацію, відслідковуваність та зберігається в приватній системі блокчейн. Технологія запобігає фальсифікації сертифікатів та дає можливість компанії повідомлювати про свої сертифікати прозорим та безпечним способом. Серед головних переваг користування технологією блокчейн компанія зазначає «аутенчичність (підтвердження істинності), децентралізацію та кодування». Також з вересня 2017 року компанія оснащує всі випущені та перевипущені сертифікати QR-кодами, а бажаючий за допомогою мобільного телефону може перевірити його достовірність, відсканувавши код, а система перевірить сертифікат у мережі блокчейн [18].

Також компанія DNV GL розробляє проект «Blockchains in the shipping world» (блокчейн у світі поставок), який включає наступні модулі: складування, відвантаження, митниця, страхування, оплата [19].

DNV GL пояснюють функцію кожного модуля наступним чином:

Модуль «Складування»: кожна одиниця товару чи відвантаження, кожна одиниця інвентарю чи документ може бути відслідкований покроково. Причини втрат чи псування можуть бути ідентифіковані. Документообіг, затримки та людські помилки вражаюче знижуються. Особливий ефект дає в поєднання з Інтернетом речей.

Модуль «Відвантаження»: особлива увага в системі менеджменту ланцюгів поставок приділяється спрощенню, зниженню документообороту та уникненню шахрайству. Відвантажуюча сторона зберігає повний контроль, інші зацікавлені сторони мають доступ лише до даних, які потрібні для їх роботи. Це значно знижує величезний обсяг супутнього документообороту та вартість транзакцій з залученням багатьох учасників. В ідеалі кожна зацікавлена сторона (судновласник, уряд, регулятивні органи, агенства) нададуть свою ноду, де відповідний блок з належною інформацією буде дублюватись. Перші випробування разом з Maersk та IBM в лютому 2017 року показали високу ефективність.

Модуль «Митниця»: митні органи повинні впевнитись в цілісності та захищеності імпортних та експортних операцій. Технологія блокчейн здатна надзвичайно сильно скоротити час та зусилля моніторингу та відслідковування відвантажень. Оформлені митні декларації, інтегровані з системою блокчейн, видимі всім зацікавленим сторонам миттєво. Імпортери матимуть достовірну інформацію про відвантаження, товар, країну походження, місце, час замитнення та всі відповідні фінансові операції. Всі відомі методи шахрайства відсікаються. Значно покращується взаємодія між державними органами, учасниками бондових, ф'ючерсних та акредитивних договорів.

Модуль «Страхування»: за рахунок миттєвої доступності та видимості всім відповідним зацікавленим сторонам значно знижуються ризики та час, необхідний на перевірку інформації. Особливо важливим є відслідковування виконання та часу виконання бізнес-процесів, які є умовою здійснення певних наступних дій. Вже випробувано разом з Maersk та Microsoft та показало вражаючі результати.

Модуль «Оплата». Міжнародні банківські транзакції у міжнародній логістиці отримують значні переваги у вигляді прискорення платіжного процесу, перегляді балансів, дебету, кредиту, боргів та зобов'язань. Моментально відслідковуються дії контрагентів, угоди стають безпечними за допомогою залучення «смарт контрактів».

9 серпня 2018 року найбільша судоходна компанія на планеті Maersk та IT гігант IBM презентували [20] свою спільну розробку TradeLens [21] – комплексну платформу для контролю відвантажень та відслідковування суден та вантажів на базі технології блокчейн, побудовану на відк-

ритих стандартах. На момент анонсування 94 організації брали активну участь або тестували платформу TradeLens. Екосистема TradeLens включала: понад 20 портів та портових операторів, що включають в себе 234 морських коридори, трьох морських перевізники, митні органи Нідерландів, Саудівської Аравії, Сінгапуру, Австралії та Перу, великі логістичні та транспорті компанії. В загальному, морські перевізники, об'єднані в систему TradeLens, покривають 20 % долі глобального ринку ланцюгів постачання [22, 23].

В травні 2019 року перевізники MSC та CMA CGM заявили про своє бажання залучитись до мережі TradeLens. Таким чином незабаром TradeLens об'єднає трьох найбільших перевізників в світі: Maersk на першому місці, MSC на другому та CMA CGM на третьому [24, 25].

Система TradeLens базується на трьох компонентах: Екосистема, Платформа та Торговий майданчик [26].

Екосистема – це, власне, бізнес-мережа учасників: відправники, порти, термінали, митниця, морські та наземні перевізники.

Платформа доступна у вигляді програмного забезпечення та поєднує екосистеми в єдине ціле за допомогою відкритих стандартів. Працює за допомогою так званого «Hyperledger Fabric blockchain» та IBM Cloud (хмарне рішення від IBM).

Торговий майданчик – відкриті додатки та сервіси дозволяють TradeLens та третім сторонам публікувати пропозиції щодо сервісів поверх Платформи.

За основу TradeLens використовує технологію блокчейн від IBM для діджиталізації ланцюга постачання, дозволяючи торговим партнерам співпрацювати, установлюючи загальне широке бачення транзакції без приватності чи конфіденційності. Відправники, морські перевізники, портові оператори, порти та термінали, наземні транспортні компанії, митні органи можуть взаємодіяти більше ефективно через доступ в реальному часі до бази даних та відправних документів, а також до Інтернету речей та даних різноманітних сенсорів широкої дії – від температурного контролю до ваги контейнера.

За допомогою блокчейн смарт-контрактів від IBM TradeLens дає можливість цифрової взаємодії багатьох сторін, що залучені до операцій міжнародної торгівлі. Модуль торгових документів, що був представлений у бета-режимі під назвою ClearWay дозволяє імпортерам/експортерам, митним брокерам та третім зацікавленим особам, таким як державні контролюючі органи, взаємодіяти в міжорганізаційних бізнес-процесах.

Протягом 12-місячного терміну випробування Maersk та IBM працювали з десятками екосистем партнерів для ідентифікації можливостей запобігти запізненням, спричинених помилками в документації, затримками у поданні інформації та інше. Один приклад продемонстрував як TradeLens зміг скоротити транзитний час поставки товару в США на 40 %, зекономивши тисячі доларів втрат. Інші приклади показали, що TradeLens змогла скоротити операційні проміжні дії, щоб відповісти на типове питання клієнта «де мій контейнер?» з десяти кроків та п'ять осіб до одного кроку та однієї особи.

Як альтернативу TradeLens та як його основний конкурент у кінці 2018 року створено подібний проект під назвою Global Shipping Business Network (GSBN) [27, 28], ще на початку процесу тестування включав дев'ять великих морських перевізників, випробування проводились на базі порту Роттердам, розробкою програмного забезпечення на основі технології блокчейн закритого типу займалась IT-компанія з Гонконку CargoSmart, яка є структурним підрозділом OOCL, створеним спеціально для цієї цілі.

Функціонал та технологія GSBN схожі до TradeLens, проте в засобах масової інформації не розголошуються досить широко [29]. Зазначається, що CargoSmart залучить свій багаж знань по доменам вантажоперевезень, аналізатори великих баз даних та багаторічний досвід розробок програмного забезпечення з штучним інтелектом, Інтернетом речей та блокчейн технологіями для того, щоб допомогти

мережі учасників покращити свої логістичні операції.

Було заявлено, що на етапі тестування нового програмного забезпечення воно буде працювати з документооборотом та перевезеннями небезпечних вантажів з метою спрощення процесу обміну інформацією з регулятивними органами та прискорення процесу відвантажень.

Проте, як зазначалось раніше, в травні 2019 року CMA CGM заявила про своє бажання залишити проект Global Shipping Business Network та долучитись до проекту TradeLens. Вірогідною причиною є повільні темпи розробки та перенесення старту початку тестової роботи платформи (таблиця).

Ще одним інноваційним проектом є проект Xeneta від американського розробника Aberdeen Group [31]. Проект фокусується на відслідковування в реальному часі вартості морського фрахту заданого маршруту від різних перевізників, їх порівняння та обрання рекомендованого перевізника (комбінації перевізників з перевантаженням в портах транзиту) з критерієм вартості та/або часу. Хоч розробники і заявили про намір виносу бази даних у мережу блокчейн, проте на сьогодні дана технологія ще не використовується.

Технологія блокчейн також була застосована для створення системи єдиної торгівлі електроенергією між торговими біржами. Компанія Волт Маркетс, що є продавцем відновлюваної енергії, застосував відкритий блокчейн для підтвердження надійності продаж та надав можливість відслідковувати сертифікати відновлюваної енергії. В лютому 2017 інший торговий дім «Меркурія» застосували таку ж технологію в сферу торгівлі нафтою з банками ING та Société Générale banks [32].

Існує декілька інших можливих варіантів застосування технології блокчейн у міжнародній логістиці, в основному щодо питання вирішення регулювання документообігу та власності на товар. Блокчейн може вирішити питання діджіталізації бортових коносаментів, основного документу при міжнародних морських перевезеннях. Бортовий коносамент – це підтвердження прийняття на борт вантажу перевізником та підтвердження права власності. Отримання цього документу часто затягується через банківські процедури, що призводить до того, що корабель перевізника прибуває у порт призначення до моменту отримання покупцем бортового коносаменту.

Ізраїльський проект Wave сфокусований на створенні «безпаперової» торгівлі, з створенням усіх транспортних документів (бортовий коносамент, інвойс, сертифікат походження, сертифікат відповідності, банківський акредитив) в електронному виді та подальшому зберіганні за технологією блокчейн. Їх головним конкурентом є американська фірма Skuchain, що знаходиться в Каліфорнії.

Шведська компанія SkyCell створила спеціальний блокчейн – асоційований контейнер з температурним режимом для повітряних перевезень біофармацевничних препаратів. Як відомо, даний тип продукції

Таблиця. Порівняння кількості учасників платформ TradeLens та Global Shipping Business Network станом на 22 квітня 2019 року [30]

TradeLens		Global Shipping Business Network	
Перевізник	Кількість контейнерів	Перевізник	Кількість контейнерів
Maersk	4,115,597	COSCO	2,822,551
PIL	405,870	CMA CGM	2,661,799
ZIM	305,247	Evergreen	1,248,375
Seaboard Marine	35,708	Yang Ming	673,354
Всього:	**4,862,422**		**7,406,079**

особливо чутливий до перепадів температур, тому контейнер здатен підтримувати температурний режим з відхиленням до 0,1 % від заданого, однак розміри такого контейнера дуже малі та підходять лише для перевезень специфічної продукції авіаційним транспортом [33].

Американська асоціація вантажних перевізників BiTA створила на базі технології блокчейн мобільний додаток для пошуку клієнтів та вантажів, який покриває 85 % наземних вантажних перевезень в США та Канаді [34].

Microsoft також активно підключились до тенденцій використання блокчейн. В партнерсві з розробником програмного та апаратного забезпечення для відслідковування вантажів та транспорту Adents розробили платформу Adents NovaTrack, яка дозволяє побачити увесь шлях руху товарів та місцезнаходження протягом всього ланцюжка постачання. Початково платформа розроблялась з метою боротьби з підробками медичних препаратів та вакцин, але пізніше розробники розширили можливу сферу застосування технології [35].

Існують також ряд проектів від менш відомих розробників, які покликані спростити взаємодію між учасниками міжнародної торгівлі та логістики, коротко розглянемо найбільш перспективні з них. Розробники Citizens Reserve анонсували запуск закритого блокчейн протоколу «Zerv», який покликаний стати «операційною системою для ланцюга постачання». Мережа, доступ до якої забезпечується токенами з реальним фінансовим забезпеченням, створена підтримувати стабільні безризикові транзакції між ключовими учасниками в ланцюгу постачання [36]. Автоматизацією комерційних процесів та розрахунків зайнялись розробники компанії Libelli. Вони розробили систему блокчейн, яка виступає в ролі банківського агента-посередника між продавцем та покупцем. Система створює систему смарт-контрактів згідно побажань учасників угоди та є альтернативою паперового банкового акредитиву. Libelli заявляють, що їх система виконує ту ж роботу, але в десять раз дешевше, ніж банківська установа. Також вони мають намір добавити функції відслідковування транспортного шляху, походження товару, аукціонні торги та інше, чим не займається банк-посередник [37]. Інший проект під назвою OriginTrail займається наданням надійної та перевіреної інформації учасникам ланцюжку постачання. Ланцюг постачання часто представляється, як один з найефективніших способів застосування технології блокчейн, але розробники заявляють, що поточні децентралізовані рішення не здатні забезпечити відповідний рівень взаємодії, стабільності та ефективності. Заявлено, що їх протокол OriginTrail відповідає високим вимогам децентралізованої мережі [38]. Варто також згадати проект ShipChain, ще одну систему відслідковування вантажів, що опирається на блокчейн. Система відслідковує товар з моменту, коли він покидає конвеєр фабрики, та до його доставки до дверей покупця. Відповідна інформація продукується на всіх етапах ланцюга постачання. Система може виконувати смарт-контракти, як тільки виконуються задані вимоги. Для прикладу, тільки водій з логістичної компанії підтверджує доставку товару покупцю, система автоматично маркує продукцію та завдання, як виконане, та надсилає новий маршрут водієві [39].

3. Смарт-контракти та мова Solidity

Смарт-контракт (англ. Smart contract – “розумний контракт”) – різновид угоди в формі закодованих математичних алгоритмів, укладення, зміна, виконання і розірвання яких можливо лише з використанням комп'ютерних програм (Блокчейн платформ) в рамках мережі Інтернет. Можна сказати, що йдеться про врегулювання відносин сторін шляхом закріплення їх вираженої волі у формі певного коду, який придатний для зчитування комп'ютером. Смарт-контракт базується на основі чіткої логіки і перевірки, або виконання через криптографічні протоколи та інші механізми цифрової безпеки, являє собою різке поліпшення в порівнянні з традиційним контрактом, навіть для деяких традиційних видів договірних положень, які можуть

бути передані під владу комп'ютерних протоколів [40, 41].

Кілька формальних мов були розроблені та запропоновані для визначення договірних положень. [42, 43].

Для розуміння поняття «смарт-контракт», що діє в мережі блокчейн «Ethereum» необхідно спочатку згадати біткоін – першу та все ще найпопулярнішу відкриту систему блокчейн. Блокчейн біткоін був розроблений з єдиною метою: передавання валюти біткоін від одного власника до іншого. Однак з поширенням технології люди почали додавати «метадані» в процесі транзакцій для досягнення інших цілей, таких як завірення та коригування документів.

Система блокчейн Ethereum виникла дещо пізніше та на відміну від інших систем блокчейн Ethereum дає змогу будь-кому створювати «контракти» всередині блокчейн. Цей контракт є комп'ютерною програмою з асоційованою мініатюрною базою даних, яка може бути змінена лише програмою, яка нею володіє. Якщо користувачі блокчейн бажають внести зміни до бази даних, вони повинні вислати повідомлення з цифровим підписом до цього контракту. Код контракту перевіряє ці повідомлення та вирішує як реагувати [44].

Контракти в Ethereum можуть бути написані на одній з декількох мов програмування, найпопулярнішими з яких є Solidity та Serpent. Як і більшість мов програмування, вони відповідають критерію «повноти за Тюрингом», що також означає, що вони мають кільцеву структуру, тобто виконують операції повторно до того часу, коли виконуються задані умови.

Коли в блокчейн створюється Ethereum-контракт, він визначає початкові дані та стан бази даних. Потім контракт призупиняє роботу, очікуючи поки користувач блокчейн (або інший контракт) відправляє повідомлення про транзакцію, після чого контракт знову починає працювати. В залежності від прописаного коду він може ідентифікувати джерело повідомлення, бути тригером для іншого контракту, змінювати базу даних чи надсилати відповідь на запит користувача. Всі ці кроки виконуються незалежно на будь-якій ноді в мережі блокчейн з ідентичними результатами.

Виконання Ethereum-контрактів вимагає певних комп'ютерних потужностей для проведення розрахунків, обчислень та зберігання інформації в мережі. Дана проблема вирішується комісією від клієнта. Користувач, який створює транзакцію, платить за комп'ютерні обчислення, які є тригером контракту, цю комісію отримує «майнер», що підтверджує інформацію та заносить її в блок. Комісія утримується покроково згідно етапам виконання контракту.

Solidity – одна з чотирьох мов (три інші: Serpent, LLL і Mutan), спроектованих для трансляції в байт-код віртуальної машини Ethereum. Отримала широке поширення з появою технологій блокчейну, зокрема, стека технологій на основі Ethereum, для створення програмного забезпечення розумних контрактів. Мова була запропонована в серпні 2014 року Гейвіном Вудом (Gavin Wood). Надалі розробка мови була виконана під керівництвом Крістіана Райтвизнера (Christian Reitwiessner) командою Solidity в рамках проекту Ethereum [45].

Solidity – об'єктно-орієнтована та предметно-орієнтована мова програмування високого рівня для виконання розумних контрактів, які керують поведінкою грошових рахунків всередині платформи Ethereum [46]. Мова Solidity була створена під впливом C++, Python та JavaScript та розроблена з ціллю взаємодії з віртуальною машиною Ethereum (Ethereum Virtual Machine або EVM). Solidity – статично типізована мова, що підтримує спадкування, бібліотеки та комплексну типізацію, визначену користувачем. На мові Solidity можна створювати контракти з функцією голосування, колективного фінансування (краудфандинг), сліпі аукціони та гаманці з мульти-підписами.

Хоч мова Solidity виглядає дещо схоже на JavaScript та C++, проте вона має ряд синтаксичних додатків, щоб бути придатною та зручною для написання контрактів в Ethereum.

4. Проблеми, які зустрічаються в ланцюгу постачання

Розглянемо типові проблеми, що зустрічаються при перевезенні товарів у міжнародній логістиці та які впливають на якість товарів, та придатність до їх подальшого використання:

1) дотримання відповідного температурного режиму. Деяким товарам (заморожена риба та м'ясо) необхідна досить низька температура (–18–20^{o}) для підтримання їх в стані глибокої заморозки, іншим товарам (свіжі овочі, фрукти, живі рослини, квіти) необхідна стабільна помірна температура (+15–20^{o}) для підтримання їх у свіжому стані. Навіть короткочасне коливання температури в заданому режимі може негативно позначитись на їх якості або навіть привести до повної втрати товару;

2) дотримання відповідної вологості повітря. Відомо, що ряд товарів бояться високої вологості або різких її перепадів. Такими товарами, для прикладу, є електротехніка, медикаменти, ряд продовольчих товарів, таких як борошно, крупи та інше. Зазвичай прості вантажні контейнери є досить герметичними, щоб захищати товар від надмірного зволоження, також у контейнер в різних місцях закладається спеціальний силікатний гель, який вбирає надлишки вологи. Проте при тривалих морських та океанічних перевезеннях судно перетинає декілька кліматичних зон, потрапляє під зливи та й просто тривалий час перебуває під впливом вологого морського повітря;

3) термін придатності продукту та його стан до початку транспортування. При дотриманні оптимальних умов транспортування товарів вони можуть витримати тривалий час міжнародних перевезень, який може сягнути двох місяців і навіть більше. Однак слід враховувати стан товарів до початку завантаження та час їх зберігання до моменту продажу або підтвердження угоди. Цей контроль досягається шляхом ветеринарного, фіто-санітарного, епідеміологічного та іншого контролю, який виконується інспекторами відповідних державних органів під час процедур імпортного та експортного митного оформлення. Тому необхідно забезпечувати швидкий та безперешкодних доступ до цих документів усім зацікавленим сторонам. Для прикладу розглянемо, як Walmart, мережа супермаркетів в США, застосували технологію блокчейн з метою контролю повного ланцюжку постачання продуктів харчування. В кооперації з IBM вони почали роботу над пілотним проектом, який створений з метою посилення прозорості ланцюга постачання та більш ефективного відслідковування шляху товару, після випадку масового отруєння забрудненим листовим салатом. З системою блокчейн, над якою вони нині працюють, було б можливо миттєво відслідкувати всі партії забрудненого салату, швидко їх прибрати з полиць та блокувати постачання від цього виробника [47]. Нещодавно Walmart провели відкрите публічне тестування відслідковування повного шляху постачання манго з Мексики та замороженої свинини з Китаю [48];

4) можливість відслідковування повного ланцюжку постачання товарів з моменту їх виробництва та до моменту отримання кінцевим споживачем. Імпортер, покупець, державні органи країни-отримувача повинні бачити увесь ланцюжок руху товарів та країни їх транзиту з метою відповідного контролю у разі транзиту вантажу через зони можливої небезпеки. Яскравим прикладом служить нещодавні спалахи епідемії африканської свинячої чуми в Україні та ряду сусідніх країн, що призвело до створення карантинних зон та заборони імпорту чи експорту свинячого м’яса з цих регіонів з метою локалізації регіонів поширення епідемії [49, 50]. Однак недобросовісні продавці шляхом транзиту через треті країни та заміну документів через посередників змогли обходити ці заборони та обмеження. Частими також є випадки завезення незвичних для даного регіону шкідників з країн транзиту, в результаті ці шкідники через відсутність природніх ворогів швидко призводили до великих втрат урожаю [51, 52]. Тому зробити увесь ланцюжок постачання товарів максимально прозорим, відкритим та доступним для контро-

люючих державних органів є вкрай важливо. Можливо навіть корегувати маршрут для уникнення зон карантину та епідемій у реальному часі;

5) можливість відслідковувати місцезнаходження вантажу в реальному часі. Для планування оплат, обміну дозвільною документацією, відповідного державного контролю необхідно знати максимально точний час прибуття вантажу в країну призначення. Сучасні технології GPS та навігації дають змогу це зробити, проте доступ до цієї інформації повинні мати не лише логістичні оператори та перевізники, а й усі зацікавлені особи та органи. Однією з перших почала звертати увагу на цю проблему відома китайська торгова онлайн платформа Alibaba – через свою дочірню компанію Lynx International, Alibaba застосували технологію блокчейн для відслідковування інформації в міжнародній логістиці. Нині система робить записи специфікації товару, деталі транспортування, митниці, державних інспекцій. Як вони самі зазначили, технологія блокчейн стала для них ідеальним рішенням [47];

6) легкість документообігу та підтвердження оригінальності документів та країни походження вантажів. Лише два десятиліття назад документообіг був повністю паперовим, що призводило до величезних втрат часу на створення документів, перевірку їх оригінальності та пересилання поштою оригіналів зацікавленим особам та державним органам контролю. Сучасні технології електронного документообігу не лише значно скорочують час та полегшують створення документів, але й значно знижують можливість підробки або заміну документів, що було частим явищем в епоху паперових документів. Технологія блокчейн та смарт-контрактів взагалі зводить можливість підробки чи корегування документів шахраями практично до нуля;

7) аварії та інші надзвичайні ситуації в морі, що призводять до псування чи втрати вантажу або значної затримки часу постачання вантажу внаслідок арешту судна до остаточного рішення суду, міжнародного морського арбітражу та реакції страхових компаній. Відомо багато випадків, коли через велику кількість учасників та постраждалих від морських аварій суд затягується на багато років. Наприклад, всесвітньо відомою є аварія в Чорному морі комерційного контейнерного танкера CMA CGM Verlaine, що перевозив близько 8700 контейнерів різного типу на борту, та круїзного судна Odessa Star, в результаті чого значна частина вантажу була втрачена, інша частина значно пошкоджена, а контейнери з температурним режимом були відключені від мережі енергопостачання на значний час, крім того для утримання судна на плаву та запобігання втрати всього вантажу команда була вимушена викинути певну частину контейнерів за борт [53, 54]. Судові слухання почалися в жовтні 2010 року, однак станом на вересень 2019 року ситуація все ще остаточно не врегульована. Час судових слухань значно б скоротився за умови достатнього обсягу достовірної та перевіреної інформації про маршрути суден, їх дії безпосередньо до та під час аварій, а також за наявності в контейнерах датчиків, що збирають та передають інформацію про стан товару;

8) своєчасна оплата за товар та гарантії безпеки продавця та покупця. Існує багато варіантів та умов оплати за товар в залежності від стану його готовності та етапу постачання, однак продавець та покупець завжди в тій чи іншій мірі несуть ризик втрати вантажу або коштів. Дана проблема вирішується шляхом механізму банківського акредитиву, коли банк-посередник виступає гарантом здійснення угоди. Класична схема акредитиву виглядає так: банк-гарант утримує кошти покупця як передплату та передає ці кошти або частину коштів поетапно продавцю за умови виконання ним певних умов, зазначених в контракті, наприклад, замитнення товару, приймання на борт судна, висилання оригінальних документів, прибуття в порт призначення та інше. Однак дана класична схема має ряд значних недоліків та незручностей: банк, як фінансовий посередник, утримує значну комісію за свої послуги, дана операція сильно розтягнута

в часі через пересилання паперових документів поштою, а також покупець у виді передплати за товар заморожує кошти на значний період часу. Згідно досліджень [55] щодня в світі у секторі міжнародної логістики заморожується 140 мільярдів доларів США через питання оплати, а також у середньому компанія змушена чекати 42 дні до моменту отримання оплати;

9) затримки в строках постачання. Згідно спільних досліджень компаній IBM та Maersk [56] в середньому простий контейнер з температурним режимом проходить через 30+ різних інстанцій та організацій, що вимагає 200+ комунікаційних зусиль. Будь-яка затримка на будь-якому етапі призводить до зриву строків поставок, а в деяких випадках навіть до втрати вантажу. Також раніше вже зазначалось про черги в доках портів для вивантаження суден;

10) пошук клієнтів/вантажів або перевізників/транспорту. Дана проблема на даний час не є критичною та практично вирішена, тому що, як зазначалося вище, на базі технології блокчейн вже створено безліч платформ та служб, що допомагають клієнтам та перевізникам знайти один одного та контактувати напряму [57].

Для уникнення або мінімізації наслідків зазначених проблем вимагається повний та безперервний контроль за станом заданих параметрів на будь-якому відрізку ланцюга постачання, особливо на стадіях перевантаження між різними логістичними операторами, наприклад, між сухопутним та морським транспортом або перевантаження контейнеру з судна в порт. На короткий час контейнер відключається від енергопостачання, тому надзвичайно важливо стежити за заданими параметрами в цей період та відстежувати можливі критичні відхилення від оптимального стану та тривалість такого періоду. Для прикладу, мінімальна температура зберігання м'яса або риби глибокої заморозки становить –18^{o}C [58], при підготовці до тривалого за часом перевантаженню доречно знизити температуру до –20–22^{o}C, що дає змогу утримувати температуру в оптимальному діапазоні протягом 15–20 годин за умови, що герметичний контейнер-рефрижиратор не буде відкриватися. Зазвичай цього часу досить, щоб перевантажити контейнер та підключити його до іншої мережі енергопостачання. Однак при мультимодальних міжнародних перевезеннях, коли використовується багато логістичних операторів та перевантажень (сухопутна доставка в країні імпорту та експорту, один або більше морських перевізників, портовий оператор у країні імпорту та експорту) вкрай важливо мати безперервний запис станів заданих параметрів якості товару з метою контролю та виявлення випадків порушень цих параметрів, а також чіткого визначення часу та місця таких порушень, логістичних операторів, винних в цьому.

Технологія блокчейн та смарт-контрактів якщо не повністю вирішують зазначені тут проблеми, то значно мінімізують їх негативний вплив, особливо ефективно в поєднанні з технологіями GPS та програмованими датчиками, що розміщуються всередині контейнерів та контролюють визначені користувачем параметри.

5. Формальна верифікація властивостей перевезень в логістиці

Системи міжнародних перевезень із часом стають складнішими і користувачу логістичних застосунків стає все важче відслідковувати та передбачати проблеми, пов'язані із перевезенням. Великі обсяги даних ускладнюють і автоматичну обробку можливих сценаріїв, оскільки їх кількість росте експоненційно при вирішуванні задач аналізу дотримування умов або властивостей ланцюжка перевезення.

Дану проблему можна вирішувати впровадженням формальних алгебраїчних методів з метою аналізу та дослідження властивостей перевезень. З цією метою в сучасній методології існують такі напрями, як перевірка моделей (model checking), що базується на використанні методів моделювання та статичного доведення властивостей. Метод широко використовується для верифікації формальних моделей, у тому числі логістичних, що реалізують деякий ланцюг постачання. Такі інструме-

нти, як MDIST [59], SPIN [60], MaceMC [61], TLC [62] показали свою практичність в застосуванні до певних індустріальних областей.

Певна складність існує у формалізації моделей, тому що найбільш розвинуті та розширені формальні методи верифікації використовують складну математичну нотацію, наприклад, системи Coq [63] та TLA.

При аналізі безпеки перевезень у моделях також проводиться формальне доведення властивостей безпеки, при чому використовуються логіки вищих порядків та автоматична система доведення теорем [64].

Логістичні моделі належать до розподілених систем, в яких бере участь деяка кількість агентів, що взаємодіють між собою. У верифікації та тестуванні розподілених систем виникає явище експоненційного вибуху, спричиненого паралельною активністю агентів, що завдає труднощі в аналізі сценаріїв логістики.

Логістичні моделі належать також до класу систем, що критичні до безпеки, тому аналіз та доведення їх властивостей стає одною із основних задач, що має вирішуватись ще на ранніх стадіях розробки моделей.

Для оцінки безпеки в логістиці використовуються так звані блок діаграми надійності (Reliability Block Diagrams) [65], які представляють ланцюг постачання. Здебільшого безпека перевезень аналізується за допомогою моделювання.

При розгляданні децентралізованої моделі логістики складність доведень зростає, тому моделювання є недостатнім і для верифікації моделі створеної на основі блокчейну використовують алгебраїчний підхід.

В Інституті кібернетики імені В.М. Глушкова НАН України розроблено теорію інсерційного моделювання [66], об'єктом якої є взаємодія агентів та середовищ. Кожен агент має свій тип, що визначається атрибутами та може взаємодіяти із іншими агентами за допомогою відправки повідомлень один одному. Агент має також свою поведінку, яка складається із дій агента та інших поведінок. Кожна дія агента визначається деякою передумовою, що дозволяє її виконання та зміною середовища агента. Середовище є сутністю над агентами, що вважаються зануреними в середовище, та взаємодіють із середовищем зміною атрибутів. Середовище має свої атрибути, які відомі всім агентам та може бути багаторівневим.

Модель поведінки агентів та середовища задається за допомогою алгебри поведінок [67], операції якої визначені над діями та поведінками агентів. Такими операціями є префіксінг, недетермінований вибір, паралельна та послідовна композиція поведінок.

Розроблені методи в алгебрі поведінок дозволяють вирішувати задачі верифікації властивостей розподілених систем такі як аналіз повноти та детермінізму, досяжність порушень формалізованих властивостей безпеки, аналіз життєздатності системи, пошук вразливостей та тестування. Блокчейн є розподіленою системою в якій взаємодіючі вузли є агентами, тому розглянуті алгебраїчні методи можуть бути застосовані до логістичної децентралізованої моделі.

Висновки

Отже, підсумовуючи все вищесказане, ми виділили глобальні проблеми, які накопичились у сучасній системі міжнародної логістики, та які можуть бути вирішені за допомогою застосування технології блокчейн та формальних методів:

- децентралізація та відмова від використання потужних стаціонарних серверів, які накопичують та зберігають величезні обсяги інформації;
- безпека зберігання та обміну інформацією. Частково вирішується децентралізацією, мінімізуючи загрози пошкодження чи збою у роботі стаціонарних серверів шляхом винесення інформації у розподілені системи, такі як блокчейн, де інформація дублюється на багатьох носіях (нодах). Також безпека досягається шляхом кодування та шифрування інформації, та можливістю доступу до неї тільки тим особам, які мають до неї безпосереднє відношення. Ще однією великою перевагою є неможливість фізичного втручання людини до попередньо внесеної інформації та

зміни нею цієї інформації із зловмисних чи інших мотивів, так як технологія блокчейн аналізує та валідує будь-які зміни інформації, що вносяться до розподіленої бази даних;

- доступність – клієнт з мінімальним рівнем технічних знань має можливість здійснювати необхідні йому операції за допомогою комп'ютера чи мобільного телефону за допомогою спеціальних додатків та програм;
- швидкодія – розподілені системи прискорюють реакцію робочої екосистеми на запит оператора, ніж стаціонарний сервер, який у часи максимального навантаження може досить довго обробляти інформацію та відповідати на запит оператора;
- економія часу та людських ресурсів – розподілені системи обробляють інформацію та відповідно реагують на неї у тисячі разів швидше, ніж людина, що дозволяє економити час та людські ресурси;
- усунення людського фактору. Більшість затримок у роботі системи та, як наслідок, втрат часу та ресурсів відбувається через людський фактор (неправильне трактування інформації, пізня реакція на збій, ігнорування сигналів системи, низька кваліфікація, природня фізична обмеженість швидкості реакції людини та аналіз інформації тощо), тому доцільно намагатися максимально усунути залежність системи від людського фактору.

Розглянувши основні проблеми в ланцюжку постачання та контролю якості товарів у міжнародній логістиці, ми бачимо величезний потенціал розвитку та застосування технології блокчейн, смарт-контрактів на її основі, які вирішують або значно спрощують ці проблеми:

- можливість постійного та безперервного контролю дотримання заданих параметрів, наприклад, температурного режиму та вологості, відслідковування порушень та чітке визначення часу, місця та етапу такого порушення;
- можливість відслідковування повного шляху товару в ланцюжку постачання, тобто від дверей виробника до дверей покупця;
- прискорення документообігу, спрощення порядку створення, верифікації документів, підвищення рівня надійності, підтвердження оригінальності документів та інформації.
- відслідковування великої кількості об'єктів логістики в реальному часі, корегування оптимального маршруту, пошук найближчого та найбільш відповідного характеристикам вантажу транспортного засобу, можливість отримання незалежної правдивої інформації про дії логістичних операторів під час колізій, аварій та інших небажаних випадків;
- зростання надійності та безпеки міжнародних торгових операцій, строків постачань та оплати через використання смарт-контрактів, прискорення розрахунків та виконання умов контрактів.

Література

1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System (2008).
2. Crosby M., Pattanayak P., Verma S., Kalyanaraman V. Blockchain technology: beyond bitcoin. Appl. Innov. 2016. 2(6). P. 6–10.
3. McPhee C., Ljutic A. Blockchain. Technol. Innov. Manag. Rev. 2017. 7(10). P. 3–5.
4. Electricity journal homepage: blockchain technology: will it make a difference? Electr. J. 2017. 30 (3). P. 86–87.
5. Finextra: Marine Transport International Applies Blockchain to Shipping Supply Chain (2016). URL: https://www.finextra.com/pressarticle/66223/marine-transportinternational-applies-blockchain-to-shipping-supply-chain
6. Solesvik M.Z. Interfirm collaboration in the shipbuilding industry: the shipbuilding cycle perspective. Int. J. Bus. Syst. Res. 2011. 5(4). P. 388–405.
7. Official website of International Maritime Organization. URL: http://www.imo.org.
8. Den Norske Veritas – DNV GL: Making your Asset Smarter with the Digital Twin. URL: https://www.dnvgl.com/article/making-your-asset-smarter-with-the-digital-twin-63328.
9. The Economist. The data deluge, 25 Feb 2010. URL: http://www.economist.com/node/15579717.

10. Tien J.M.: Internet of connected servgoods: considerations, consequences and concerns. J. Syst. Sci. Syst. Eng. 2015. 24(2). P. 130–167.
11. Splash 24: Dr. Martin Stopford on the Future of Shipping. URL: http://splash247.com/dr-martinstopford-future-shipping
12. Kondratenko Y.P., Kozlov O.V., Korobko O.V., Topalov A.M. Internet of things approach for automation of the complex industrial systems. In: ICTERI-2017, CEUR Workshop Proceedings Open Access. 2017. Vol. 1844. P. 3–18. URL: https://pdfs.semanticscholar.org/3ff6/4e4a07be1e8c2f0b16f4736397be1405218a.pdf
13. Tachizawa E.M., Alvarez-Gil M.J., Montes-Sancho M.J. How "smart cities" will change supply chain management. Supply Chain Manag. Int. J. 2015. 20(3). P. 237–248.
14. Hahn G.J., Packowski J. A perspective on applications of in-memory analytics in supply chain management. Decis. Support Syst. 2015. 76(1). P. 45–52.
15. Lacey M., Lisachuk H., Giannopoulos A., Ogura A. Shipping smarter: IoT opportunities in transport and logistics. The Internet of Things in Shipping. Dupress-Deloitte. 2015.
16. Offshore Energy Today: Meet Onboard, the Maritime Internet of Things. URL: http://www.offshoreenergytoday.com/oeec-meet-onboard-the-maritime-internet-of-things/?utm_source=emark&utm_medium=email&utm_campaign=daily-update-offshore-energy-today-2017-10-02&uid=207087.
17. Maritime: Six Maritime Start-Ups That Are Changing the GAME. URL: https://knect365.com/talentandtraining/article/1149354e-68d9-4e74-9f91-a900ac869526/6-maritime-startupsthat-are-changing-the-game.
18. Den Norske Veritas – DNV GL: Certificates in the blockchain. URL: https://www.dnvgl.com/assurance/certificates-in-the-blockchain.html.
19. Den Norske Veritas – DNV GL: Blockchains in the shipping world. URL: https://www.dnvgl.com/expert-story/maritime-impact/Blockchains-in-the-shipping-world.html.
20. Maersk and IBM Introduce TradeLens Blockchain Shipping Solution. Aug. 9, 2018. URL: https://newsroom.ibm.com/2018-08-09-Maersk-and-IBM-Introduce-TradeLens-Blockchain-Shipping-Solution.
21. TradeLens – official website. URL: https://www.tradelens.com.
22. Forbes. IBM-Maersk Blockchain Platform Adds 92 Clients As Part Of Global Launch. URL: https://www.forbes.com/sites/michael-delcastillo/2018/08/09/ibm-maersk-blockchain-platform-adds-92-clients-as-part-of-global-launch-1/#c469d5068a4a.
23. Coindesk. IBM and Maersk Struggle to Sign Partners to Shipping Blockchain. Oct 26, 2018. URL: https://www.coindesk.com/ibm-blockchain-maersk-shipping-struggling
24. Computerworld. Maersk adds two big shipping firms to its blockchain ledger. 29 травня 2019р. URL: https://www.computerworld.com/article/3398923/maersk-adds-two-big-shipping-firms-to-its-blockchain-ledger.html
25. TC. IBM-Maersk blockchain shipping consortium expands to include other major shipping companies. URL: https://techcrunch.com/2019/05/28/ibm-maersk-blockchain-shipping-consortium-expands-to-include-other-major-shipping-companies/
26. TradeLens official brochure. Solution brief. Edition two.
27. SupplyChain Dive. 9 ocean carriers, terminal operators join new blockchain initiative to rival TradeLens. Nov. 7, 2018. URL: https://www.supplychaindive.com/news/ocean-carriers-new-blockchain-cosco-cma-cgm/541630/
28. CargoSmart. Global Shipping Business Network. URL: https://www.cargosmart.ai/en/
29. Globe news wire. Top Ocean Carriers and Terminal Operators Initiate Blockchain Consortium. November 06, 2018. URL: https://www.globenewswire.com/news-release/2018/11/06/1646014/0/en/Top-Ocean-Carriers-and-Terminal-Operators-Initiate-Blockchain-Consortium.html
30. Supply Chain Dive. Maersk blockchain solution TradeLens adds ZIM. April 22, 2019. URL: https://www.supplychaindive.com/news/maersk-blockchain-solution-tradelens-adds-zim/553146/
31. Ball B. Reducing Global Logistics Cost with Benchmarking and Shipping Container Pricing Transparency. Aberdeen Group. 2016.
32. Dhanji T. Blockchain – Where Oil and Gas Traders Dare to Trade. Ernest Young Publications. 2017. URL: https://www.linkedin.com/pulse/blockchain-where-oilgas-traders-dare-tread-talib-dhanji
33. Freight Waves.: Swiss firm brings blockchain to the biopharmaceutical cold chain. 02/23/2018. URL: https://www.freightwaves.com/news/blockchain/skycellblockchaincoldchain.
34. John G. Smith.: Block by Block: How blockchain will transform trucking. 18.01.2018. URL: https://www.todaystruc-

king.com/block-block-blockchain-will-transform-trucking/
35. Ana Alexandre.: New Blockchain-Based Supply Chain System Is Presented by Microsoft and Adents. URL: https://cointelegraph.com/news/new-blockchain-based-supply-chain-system-is-presented-by-microsoft-and-ardents
36. Suku. The future of supply chain is here. URL: https://www.suku.world/
37. DHL Trend Research. URL: https://www.logistics.dhl/content/dam/dhl/global/core/documents/pdf/glo-core-blockchain-trend-report.pdf
38. OriginTrail. URL: https://origintrail.io
39. Shipchain. The end-to-end logistics platform of the future: trustless, transparent tracking. URL: https://shipchain.io/
40. Nick Szabo – Smart Contracts: Building Blocks for Digital Markets. URL: www.fon.hum.uva.nl
41. Nick Szabo. Formalizing and Securing Relationships on Public Networks.
42. Welcome to ERights.Org. URL: erights.org
43. A Formal Language for Analyzing Contracts. URL: vwh.net
44. Gideon Greenspan.: Smart contracts make slow blockchains. URL: https://www.multichain.com/blog/2015/11/smart-contracts-slow-blockchains
45. StackEdit Viewer / Benoit Schweblin. URL: https://stackedit.io/viewer#!url=https://gist.githubusercontent.com/gavofyork/31b35cd2252a00d0d057/raw/16de06189d2175d2e31b300f1f8531e20c927635/solidity-original
46. Solidity – Solidity 0.2.0 documentation. URL: https://solidity.readthedocs.io/en/latest/
47. Zigurat. Blockchain Success cases: Supply Chain and Logistics. URL: https://www.e-zigurat.com/innovation-school/blog/blockchain-success-cases/
48. Robert Hackett.: Walmart and 9 Food Giants Team Up on IBM Blockchain Plans. URL: https://fortune.com/2017/08/22/walmart-blockchain-ibm-food-nestle-unilever-tyson-dole/
49. Карта випадків африканської чуми свиней в Україні та зони карантину. URL: http://www.asf.vet.ua/index.php/asfinukraine
50. Державна служба України з питань безпечності харчових продуктів та захисту споживачів: Африканська чума свиней? URL: http://www.asf.vet.ua/index.php/ purpose-project/about-asf/124-african-swine-fever
51. Укрінформ. Нашестя метеликів – розплата за людську необачність. 03.06.2019 URL: https://www.ukrinform.ua/rubric-economy/2713807-nasesta-metelikiv-rozplata-za-ludsku-nenazerlivist.html
52. Petropavlivka. City. Миколаївку атакує гусінь. 05.06.2019 URL: https://petropavlivka.city/read/experiance/33845/mikolaivku-atakue-gusin
53. Carnet Maritime. Collision CMA CGM Verlaine et Odessa Star. 04.04.2010 URL: http://carnet-maritime.com/accidents-naufrages/collision-cma-cgm-verlaine-et-odessa-star.html
54. Информационный портал «Транспортный бизнес Украины». К ситуации столкновения контейнеровоза Verlaine и теплохода Odessa Star 4 апреля в Мраморном море - Международная юридическая служба. URL: http://tbu.com.ua/news/k_situatsii_stolknoveniia_konteinerovoza_verlaine_i_teplohoda_odessa_star_4_aprelia_v_mramornom_more___mejdunarodnaia_uridicheskaia_slujba_foto.html
55. Cristina Commendatore.: Blockchain in trucking: What about the middlemen? 20.10.2017 URL: https://www.fleetowner.com/electronic-security/blockchain-trucking-what-about-middlemen
56. Robert Hackett.: IBM and Maersk Are Creating a New Blockchain Company. 16.01.2017 URL: https://fortune.com/2018/01/16/ibm-blockchain-maersk-company/
57. Winnesota.: How blockchain is revolutionizing the world of transportation and logistics. URL: https://www.winnesota.com/blockchain
58. Temperature Measurement in the Fish Industry. URL: http://www.fao.org/3/x5992e/X5992e01.htm
59. Yang J., Chen T., Wu M., Xu Z., Liu X., Lin H., Yang M., Long F., Zhang L., Zhou L. 2009. MODIST: transparent model checking of unmodified distributed systems. URL: https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/yang/yang_html/index.html
60. Spin. URL: http://spinroot.com/spin/whatispin.html
61. Killian C., Anderson J.W., Jhala R., Vahdat A. 2006. Life, death, and the critical transition: finding liveness bugs in system code. URL: http://www.macesystems.org/papers/MaceMC_TR.pdf
62. Lamport L., Yu Y. 2011. TLC—the TLA+ model checker. URL: http://research.microsoft.com/en-us/um/people/lamport/tla/tlc.html

63. Wilcox J.R., Woos D., Panchekha P., Tatlock Z., Wang X., Ernst M.D., Anderson T. 2015. Verdi: a framework for implementing and formally verifying distributed systems. Proceedings of the ACM SIGPLAN 2015 Conference on Programming Language Design and Implementation: 357-368. URL: https://homes.cs.washington.edu/ ~mernst/pubs/verify-distsystem-pldi2015-abstract.html
64. Towards Formal Reliability Analysis of Logistics Service Supply Chains using Theorem Proving Waqar Ahmed , Osman Hasan , and Sofi`ene Tahar EPiC Series in Computing Volume 40, 2016, Pages 1–14 IWIL-2015. 11th International Workshop on the Implementation of Logics
65. Li Y., Yi H. Research on the Inherent Reliability and the Operational Reliability of the Supply Chain. u- and e-Service, Science and Technology. 2014. 7(1). P. 104–112.
66. Letichevsky A., Letychevskyi O., Peschanenko V. "Insertion Modeling and Its Applications", Computer Science Journal of Moldova. 2016. Vol. 24. Issue 3. P. 357–370.
67. Letichevsky A. and Gilbert D. "Interaction of agents and environments," in: Recent Trends in Algebraic Development Technique, LNCS 1827 (D. Bert and C. Choppy, eds.), Springer-Verlag, 1999.

References

1. Nakamoto S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008).
2. Crosby M., Pattanayak P., Verma S., Kalyanaraman V.: Blockchain technology: beyond bitcoin. Appl. Innov. 2016. 2(6).,P. 6–10.
3. McPhee C., Ljutic A.: Blockchain. Technol. Innov. Manag. Rev. 2017. 7(10). P. 3–5.
4. Electricity journal homepage: blockchain technology: will it make a difference? Electr. J. 2017. 30 (3). P. 86–87.
5. Finextra: Marine Transport International Applies Blockchain to Shipping Supply Chain (2016). URL: https://www.finextra.com/ pressarticle/66223/marine-transportinternatio-nal-applies-blockchain-to-shipping-supply-chain
6. Solesvik M.Z.: Interfirm collaboration in the shipbuilding industry: the shipbuilding cycle perspective. Int. J. Bus. Syst. Res. 2011. 5(4). P. 388–405.
7. Official website of International Maritime Organization. URL: http://www.imo.org.
8. Den Norske Veritas – DNV GL: Making your Asset Smarter with the Digital Twin. URL: https://www.dnvgl.com/article/making-your-asset-smarter-with-the-digital-twin-63328.
9. The Economist. The data deluge, 25 Feb 2010. URL: http://www.economist.com/ node/15579717.
10. Tien J.M.: Internet of connected servgoods: considerations, consequences and concerns. J. Syst. Sci. Syst. Eng. 2015. 24(2). P. 130–167.
11. Splash 24: Dr. Martin Stopford on the Future of Shipping. URL: http://splash247.com/dr-martinstopford-future-shipping
12. Kondratenko Y.P., Kozlov O.V., Korobko O.V., Topalov A.M.: Internet of things approach for automation of the complex industrial systems. In: ICTERI-2017, CEUR Workshop Proceedings Open Access. 2017. Vol. 1844. P.3–18. URL: https://pdfs.semanticscholar.org/3ff6/4e4a07b e1e8c2f0b16f4736397be1405218a.pdf
13. Tachizawa E.M., Alvarez-Gil M.J., Montes-Sancho M.J.: How "smart cities" will change supply chain management. Supply Chain Manag. Int. J. 2015. 20(3). P. 237–248.
14. Hahn G.J., Packowski J.: A perspective on applications of in-memory analytics in supply chain management. Decis. Support Syst. 2015. 76(1). P.45–52.
15. Lacey M., Lisachuk H., Giannopoulos A., Ogura A.: Shipping smarter: IoT opportunities in transport and logistics. The Internet of Things in Shipping. Dupress-Deloitte. 2015.
16. Offshore Energy Today: Meet Onboard, the Maritime Internet of Things. URL: http://www.offshoreenergytoday.com/oeec-meet-onboard-the-maritime-internet-of-things/?utm_source=emark&utm_medium=e mail&utm_campaign=daily-update-offshore-energy-today-2017-10-02&uid=207087.
17. Maritime: Six Maritime Start-Ups That Are Changing the GAME. URL: https://knect365.com/talentandtraining/article/ 1149354e-68d9-4e74-9f91-a900ac869526/6-maritime-startupsthat-are-changing-the-game.
18. Den Norske Veritas – DNV GL: Certificates in the blockchain. URL: https://www.dnvgl.com/ assurance/certificates-in-the-blockchain.html.
19. Den Norske Veritas – DNV GL: Blockchains in the shipping world. URL: https://www.dnvgl.com/expert-story/mariti-me-impact/Blockchains-in-the-shipping-world.html.
20. Maersk and IBM Introduce TradeLens

Blockchain Shipping Solution. Aug. 9, 2018. URL: https://newsroom.ibm.com/2018-08-09-Maersk-and-IBM-Introduce-TradeLens-Blockchain-Shipping-Solution.
21. TradeLens – official website. URL: https://www.tradelens.com.
22. Forbes. IBM-Maersk Blockchain Platform Adds 92 Clients As Part Of Global Launch. URL: https://www.forbes.com/sites/michael-delcastillo/2018/08/09/ibm-maersk-blockchain-platform-adds-92-clients-as-part-of-global-launch-1/#c469d5068a4a.
23. Coindesk. IBM and Maersk Struggle to Sign Partners to Shipping Blockchain. Oct 26, 2018. URL: https://www.coindesk.com/ibm-blockchain-maersk-shipping-struggling
24. Computerworld. Maersk adds two big shipping firms to its blockchain ledger. 29 травня 2019р. URL: https://www.computerworld.com/article/3398923/maersk-adds-two-big-shipping-firms-to-its-blockchain-ledger.html
25. TC. IBM-Maersk blockchain shipping consortium expands to include other major shipping companies. URL: https://techcrunch.com/2019/05/28/ibm-maersk-blockchain-shipping-consortium-expands-to-include-other-major-shipping-companies/
26. TradeLens official brochure. Solution brief. Edition two.
27. SupplyChain Dive. 9 ocean carriers, terminal operators join new blockchain initiative to rival TradeLens. Nov. 7, 2018. URL: https://www.supplychaindive.com/news/ocean-carriers-new-blockchain-cosco-cma-cgm/541630/
28. CargoSmart. Global Shipping Business Network. URL: https://www.cargosmart.ai/en/
29. Globe news wire. Top Ocean Carriers and Terminal Operators Initiate Blockchain Consortium. November 06, 2018. URL: https://www.globenewswire.com/news-release/2018/11/06/1646014/0/en/Top-Ocean-Carriers-and-Terminal-Operators-Initiate-Blockchain-Consortium.html
30. Supply Chain Dive. Maersk blockchain solution TradeLens adds ZIM. April 22, 2019. URL: https://www.supplychaindive.com/news/maersk-blockchain-solution-tradelens-adds-zim/553146/
31. Ball B. Reducing Global Logistics Cost with Benchmarking and Shipping Container Pricing Transparency. Aberdeen Group. 2016.
32. Dhanji T.: Blockchain – Where Oil and Gas Traders Dare to Trade. Ernest Young Publications. 2017. URL: https://www.linkedin.com/pulse/blockchain-where-oilgas-traders-dare-tread-talib-dhanji
33. Freight Waves.: Swiss firm brings blockchain to the biopharmaceutical cold chain. 02/23/2018. URL: https://www.freightwaves.com/news/blockchain/skycellblockchaincoldchain.
34. John G. Smith.: Block by Block: How blockchain will transform trucking. 18.01.2018. URL: https://www.todaystrucking.com/block-block-blockchain-will-transform-trucking/
35. Ana Alexandre.: New Blockchain-Based Supply Chain System Is Presented by Microsoft and Adents. URL: https://cointelegraph.com/news/new-blockchain-based-supply-chain-system-is-presented-by-microsoft-and-ardents
36. Suku. The future of supply chain is here. URL: https://www.suku.world/
37. DHL Trend Research. URL: https://www.logistics.dhl/content/dam/dhl/global/core/documents/pdf/glo-core-blockchain-trend-report.pdf
38. OriginTrail. URL: https://origintrail.io
39. Shipchain. The end-to-end logistics platform of the future: trustless, transparent tracking. URL: https://shipchain.io/
40. Nick Szabo – Smart Contracts: Building Blocks for Digital Markets. URL: www.fon.hum.uva.nl
41. Nick Szabo. Formalizing and Securing Relationships on Public Networks.
42. Welcome to ERights.Org. URL: erights.org
43. A Formal Language for Analyzing Contracts. URL: vwh.net
44. Gideon Greenspan.: Smart contracts make slow blockchains. URL: https://www.multichain.com/blog/2015/11/smart-contracts-slow-blockchains
45. StackEdit Viewer / Benoit Schweblin. URL: https://stackedit.io/viewer#!url=https://gist.githubusercontent.com/gavofyork/31b35cd2252a00d0d057/raw/16de06189d2175d2e31b300f1f8531e20c927635/solidity-original
46. Solidity – Solidity 0.2.0 documentation. URL: https://solidity.readthedocs.io/en/latest/
47. Zigurat. Blockchain Success cases: Supply Chain and Logistics. URL: https://www.e-zigurat.com/innovation-school/blog/blockchain-success-cases/
48. Robert Hackett.: Walmart and 9 Food Giants Team Up on IBM Blockchain Plans. URL: https://fortune.com/2017/08/22/walmart-blockchain-ibm-food-nestle-unilever-tyson-dole/

49. Map of cases of African swine fever in Ukraine and quarantine areas. [In Ukrainian] URL: http://www.asf.vet.ua/index.php/asfinukraine
50. The State Service of Ukraine for Food Safety and Consumer Protection: African swine fever? [In Ukrainian] URL: http://www.asf.vet.ua/index.php/purpose-project/about-asf/124-african-swine-fever
51. Ukrinform. The butterfly invasion is the pay for human indiscretion. 03.06.2019. [In Ukrainian] URL: https://www.ukrinform.ua/rubric-economy/2713807-nasesta-metelikiv-rozplata-za-ludsku-nenazerlivist.html
52. Petropavlivka.City. Mykolayivka is attacked by a caterpillar. 05.06.2019. [In Ukrainian] URL: https://petropavlivka.city/read/experiance/33845/mikolaivku-atakue-gusin
53. Carnet Maritime. Collision CMA CGM Verlaine et Odessa Star. 04.04.2010 URL: http://carnet-maritime.com/accidents-naufrages/collision-cma-cgm-verlaine-et-odessa-star.html
54. Information portal "Transport business of Ukraine". On the situation of collision of the container carrier Verlaine and the ship Odessa Star on April 4 in the Marmara Sea - International Legal Service. [In Ukrainian] URL: http://tbu.com.ua/news/k_situatsii_stolknoveniia_konteinerovoza_verlaine_i_teplohoda_odessa_star_4_aprelia_v_mramornom_more___mejdunarodnaia_uridicheskaia_sluj ba_foto.html
55. Cristina Commendatore.: Blockchain in trucking: What about the middlemen? 20.10.2017 URL: https://www.fleetowner.com/electronic-security/blockchain-trucking-what-about-middlemen
56. Robert Hackett.: IBM and Maersk Are Creating a New Blockchain Company. 16.01.2017 URL: https://fortune.com/2018/01/16/ibm-blockchain-maersk-company/
57. Winnesota.: How blockchain is revolutionizing the world of transportation and logistics. URL: https://www.winnesota.com/ blockchain
58. Temperature Measurement in the Fish Industry. URL: http://www.fao.org/3/x5992e/X5992e01.htm
59. Yang, J., Chen, T., Wu, M., Xu, Z., Liu, X., Lin, H., Yang, M., Long, F., Zhang, L., Zhou, L. 2009. MODIST: transparent model checking of unmodified distributed systems. URL: https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/yang/yang_html/index.html
60. Spin. URL: http://spinroot.com/spin/whatispin.html
61. Killian, C., Anderson, J. W., Jhala, R., Vahdat, A. 2006. Life, death, and the critical transition: finding liveness bugs in system code. URL: http://www.macesystems.org/papers/MaceMC_TR.pdf
62. Lamport, L., Yu, Y. 2011. TLC—the TLA+ model checker. URL: http://research.microsoft.com/en-us/um/people/lamport/tla/tlc.html
63. Wilcox J. R., Woos D., Panchekha P., Tatlock Z., Wang X., Ernst M. D., Anderson T. 2015. Verdi: a framework for implementing and formally verifying distributed systems. Proceedings of the ACM SIGPLAN 2015 Conference on Programming Language Design and Implementation: 357-368. URL: https://homes.cs.washington.edu/~mernst/pubs/verify-distsystem-pldi2015-abstract.html
64. Towards Formal Reliability Analysis of Logistics Service Supply Chains using Theorem Proving Waqar Ahmed , Osman Hasan , and Sofi`ene Tahar EPiC Series in Computing Volume 40, 2016, Pages 1–14 IWIL-2015. 11th International Workshop on the Implementation of Logics
65. Li Y., Yi H. Research on the Inherent Reliability and the Operational Reliability of the Supply Chain. u- and e-Service, Science and Technology. 2014. 7(1). P. 104–112.
66. Letichevsky A., Letychevskyi O., Peschanenko V. "Insertion Modeling and Its Applications", Computer Science Journal of Moldova. 2016. Vol. 24. Issue 3. P. 357–370.
67. Letichevsky A. and Gilbert D. "Interaction of agents and environments," in: Recent Trends in Algebraic Development Technique, LNCS 1827 (D. Bert and C. Choppy, eds.), Springer-Verlag, 1999.

Одержано 20.11.2019

Про авторів:

Летичевський Олександр Олександрович,
доктор фізико-математичних наук,
завідувач відділу теорії
цифрових автоматів.
Кількість наукових публікацій в
українських виданнях – 32.
Кількість наукових публікацій в
зарубіжних виданнях – 37.
Індекс Гірша – 4.
https://orcid.org/0000-0003-0856-9771,

Горбатюк Сергій Олександрович,
аспірант кафедри Комп'ютерних наук та інформаційних технологій.
https://orcid.org/0000-0001-6834-4211.

Місце роботи авторів:

Інститут кібернетики імені В.М. Глушкова
НАН України.
03187, м. Київ
проспект Академіка Глушкова, 40.
Тел.: (044) 526 20 08.

E-mails: gorbatiuk_sergiy@i.ua,
lit@issukraine.com

UDC 004.432 https://doi.org/10.15407/pp2020.01.074

Yevgeniya Sulema, Vladyslav Glinskii

SEMANTICS AND PRAGMATICS OF PROGRAMMING LANGUAGE ASAMPL

This paper presents semantics and practical implementation of the domain-specific programming language ASAMPL. This programming language has been developed to support the efficient processing of multimodal data processing, in particular, the processing of multimedia content which components are evidently defined in terms of time. The data processing concept employed in ASAMPL is based on the data structures, operations, and relations defined in the algebraic system of aggregates. The paper explains the compilation approach used for this programming language as well as it presents the test results and their discussion.
Key words: multimedia, multimodal data, programming language, compilation.

Introduction

Nowadays, such technologies as IoT, machine learning, big data, augmented and virtual reality are developing very fast and they are more and more in use for solving everyday tasks. For example, virtual assistants like Amazon Alexa or Apple Siri help users by employing natural language processing, voice recognition and computer vision along with other artificial intelligence techniques.

Multimedia technologies along with AR and VR allow us to make education more efficient [1–3]. Some examples are interactive classes and training programs for medical students, technicians, drivers, etc. These technologies enable modelling processes and visualize them, recreating required environment, improve skills while minimizing loss due to learner's mistake. They can be used to make monitoring and operating machines or even industrial complexes much simpler. As an example, it can greatly improve operation of the drone used in emergency situation by giving to operator visualization of the environment, identified objects or threats.

IoT solutions are used not only in industrial and agricultural complexes but also to create "smart homes" and "smart cities". Data gathered from different sensors can be used to make immediate decisions to control operation of connected equipment. At the same time this data is stored for analysis and prediction of the future trends in self-learning systems, visualization to human operators.

Rising amounts of diverse data push requirements to its storage, transmission and processing technologies further. So, nowadays we have real need to work with so-called multimodal data, i.e. data which has different origin, such as images, audio, environmental characteristics (humidity, temperature), etc. Usually such data is constantly changing in time. In some cases, the use of general-purpose programming languages for multimodal data processing tasks is not efficient enough when timewise processing is required. The understanding of this fact has leaded to the creation of a domain-specific programming language ASAMPL [4]. The purpose of the research presented in this paper is to make the semantics of ASAMPL more mature as well as to carry out the reference implementation of this language by developing a compiler and a standard library.

1. Concept and features of ASAMPL

The ASAMPL programming language derives its name from the "Algebraic System of Aggregates" and the "Mulsemedia data Processing Language" [4]. ASAMPL has been designed for simple and efficient multimodal data processing which is based on the following facts: (1) every element of such data is related to a certain point in time when it has been obtained (recorded, generated, measured, etc.); (2) multimodal data processing needs to take into account the modality of the data, its compatibility and the possible interrelation between data of different modalities. Therefore, the basic principle of organizing a program in this language is that the program code execution is governed by two entities: time and data modality.

A key concept in ASAMPL is the concept of multi-image which is based on the mathematical apparatus of the Algebraic System of Aggregates (ASA) [5-7]. To represent the multi-image of a real-world object, two basic data structures defined in ASA, namely, tuples and aggregates, are employed in this programming language. The processing of tuples and aggregates is performed in accordance with the rules prescribed in ACA.

The main features of the ASAMPL language which specify the principles of the programming paradigm for multimodal data structures processing as the following:

1. Linking the data processing process to a timeline.
2. Synchronizing the data of different modalities.
3. Complex representation of multimodal data using tuples and aggregates.
4. Focusing on the simultaneous use of various multimodal data sources (data streams from remote sensors, media files from cloud storages, etc.).
5. Dynamic linking external libraries, renders (a renderer is a software tool for reproducing the data of a certain modality), handlers for encoding / decoding, data processing and reproduction for each modality, use of specific file formats and predefined devices.
6. Mathematical processing of data aggregates based on ACA.

A program code is ASAMPL consists of blocks starting with the following keywords:

– ACTIONS, it defines data processing logic;

– AGGREGATES, it defines aggregates composed from defined tuples;

– ELEMENTS, it defines variables and their belonging to defined sets;

– HANDLERS, it defines built-in and external modules for data converting;

– LIBRARIES, it defines built-in and external libraries;

– PROGRAM, it starts a program code;

– RENDERERS, it defines built-in and external modules for data rendering;

– SETS, it defines sets which contain data elements;

– SOURCES, it defines data sources where multimodal data is stored, recorded, generated, etc.

– TUPLES, it defines tuples and their belonging to defined sets.

The program structure is as follows:

```
Program <name> {
  Libraries {
    <library1> is <path-
ToLibray1>;
    <library2> is <pathToLi-
brary2>;
    …… }
  Handlers {
    <handler1> is
<pathToHandler1>;
    <handler2> is
<pathToHandler2>;
    …… }
  Renderers {
    <renderer1> is <pathToRender-
er1>;
    <renderer2> is <pathToRender-
er2>;
    …… }
  Sources {
    <source1> is <pathToSource1>;
    <source2> is <pathToSource2>;
    …… }
  Sets {
    <set1> is <type1>;
    <set2> is <type2>;
    …… }
  Elements {
    <element1> is <set>;
    <element2> = <value>;
    …… }
  Tuples {
    <tuple1> = <set1>;
    <tuple2> = <set2>;
    …… }
  Aggregates {
    <aggregate1> = [<tuple1>, …,
<tupleM>];
    <aggregate2> = [<tupleK>, …,
<tupleN>];
    …… }
  Actions {
    <list of statements>   } }
```

The logic of multimodal data structures timewise processing is set by the operators which are based on the mathematical concept defined in ASA, in particular, logical

and ordering operations on aggregates and frequency and interval relations of discrete intervals as well as the concept of a multi-image [3–5].

To implement the multimodal data timewise processing logic, the ASAMPL specification includes the following operators: TIMELINE; SEQUENCE; SUBSTITUTE FOR WHEN; DOWNLOAD FROM WITH; UPLOAD TO WITH; RENDER WITH.

To define the semantics of the ASAMPL operators, the axiomatic approach is used in this research [8, 9]. An axiomatic specification is defined as the following derivation rule:

$$\frac{H_1, H_2, \ldots, H_n}{H},$$

where H is true if $\forall\ H_i$ is true, $i = [1..n]$.

TIMELINE operator enables timewise data processing by allowing a programmer to apply a certain action during a defined period of time. All actions included into the operator body are to be executed simultaneously. This operator has three options for the timeline defining.

The first option requires an evident indication of both the beginning and the end time moments of the timeline:

```
TIMELINE time1 : step : time2
{ list of actions to be carried
out simultaneously }
```

The semantics rule for the first option of TIMELINE operator is:

$$\frac{\{P \,\&\, (a \le x \le (a + \Delta a \cdot n))\}\, C\, \{Q\}}{\{P\}\ \text{TIMELINE}\ a : \Delta a : (a + \Delta a \cdot n)\ C\ \{Q\}}.$$

The second option allows a programmer to specify an arbitrary tuple of time values which correspond to specific time moments when the actions from the given list need to be carried out:

```
TIMELINE AS time_tuple { list of
actions to be carried out simul-
taneously }
```

The semantics rule for the second option of TIMELINE operator is:

$$\frac{\{P \,\&\, (x \in [a_1 .. a_n])\}\, C\, \{Q\}}{\{P\}\ \text{TIMELINE AS}\, [a_1 .. a_n]\ C\ \{Q\}}.$$

The third option of the TIMELINE operator enables using a condition which triggers the completion of the given actions fulfilment:

```
TIMELINE UNTIL condition
{ list of actions to be carried
out simultaneously }
```

The semantics rule for the third option of TIMELINE operator is:

$$\frac{\{P \,\&\, B\}\, C\, \{Q\}}{\{P\}\ \text{TIMELINE UNTIL}\ B\ C\ \{Q\}}.$$

SEQUENCE operator enables sequential processing of actions as one compound action:

```
SEQUENCE { list of actions to be
carried out sequentially }
```

The semantics rule for this operator is as follows:

$$\frac{\{P\}\, C_1\, \{Q\},\ \{Q\}\, C_2\, \{R\}}{\{P\}\ \text{SEQUENCE}\ C_1; C_2\ \{R\}}.$$

SUBSTITUTE FOR WHEN operator enables replacement of one data set by another one if a certain condition becomes true:

```
SUBSTITUTE name1 FOR name2
WHEN logical_expression.
```

This operator can be useful in the case when, for example, high-resolution data cannot be downloaded, uploaded, or processed because of insufficient data rates and, thus, they can be replaced with low-resolution data to allow an application to be executed even in such inappropriate conditions.

The semantics rule for this operator is as the following:

$$\frac{\{P \,\&\, B\}\, C_1\, \{Q\},\ \{P \,\&\, \overline{B}\}\, C_2\, \{Q\},\ C_i \equiv (d := d_i)}{\{P\}\ d\ \text{SUBSTITUTE}\ d_2\ \text{FOR}\ d_1\ \text{WHEN}\ B\ \{Q\}}$$

where $i = [1, 2]$.

DOWNLOAD FROM WITH operator enables downloading data from a specified data source by using either a predefined handler, or a given one. In the first case, the operator has the following format:

```
DOWNLOAD data_name FROM source
```

Its semantic rule is as follows:

$$\frac{\{P\}\, C\, \{Q\}}{\{P\}\ \text{DOWNLOAD}\ d\ \mathit{FROM}\ d_1\ \{Q\}}.$$

In the second case, the handler identifier needs to be indicated directly:

```
DOWNLOAD data_name FROM source
WITH handler
```

Its semantic rule for this case is:

$$\frac{\{P\}\, C\, \{Q\}}{\{P\}\ \text{DOWNLOAD}\ d\ \mathit{FROM}\ d_1\ \text{WITH}\ z\ \{Q\}}.$$

UPLOAD TO WITH operator allows to upload any data (an element, a tuple, an aggregate) to a defined resource by using either a predefined handler, or a given one. In the first option, there is no need to indicate the handler because it is predefined:

```
UPLOAD data_name TO destination
```

The semantic rule for the first option is the following:

$$\frac{\{P\}\, C\, \{Q\}}{\{P\}\ \text{UPLOAD}\ d\ \mathit{TO}\ d_1\ \{Q\}}.$$

In the second option, the handler needs to be indicated evidently:

```
UPLOAD data_name TO destination
WITH handler
```

The semantic rule for the second option is the following:

$$\frac{\{P\}\, C\, \{Q\}}{\{P\}\ \text{UPLOAD}\ d\ \mathit{TO}\ d_1\ \text{WITH}\ z\ \{Q\}}.$$

RENDER WITH operator enables data reproduction. Each data modality is preprocessed and rendered by its specific render. The form of this operator is as follows:

```
RENDER data_name WITH render_name
```

The semantic rule for this operator is the following:

$$\frac{\{P\}\, C\, \{Q\}}{\{P\}\ \text{RENDER}\ d\ \text{WITH}\ z\ \{Q\}}.$$

Besides, ASAMPL employs a branch statement IF THEN; a selection statement CASE OF; an assignment statement IS.

Thus, the logic of the data processing is based on these basic actions. The method of ASAMPL code translation is presented in the next section.

2. Compilation approach

The compiler implementation language needs to be strongly typed high-level general-purpose programming language with simple and expressive syntax. Automatic memory management is desired but not mandatory. Besides, this language should be popular and mature enough because popular languages tend to have bigger community, comprehensive documentation, many ready-to-use program components, frameworks and libraries. The analysis of top programming languages [10, 11] has been resulted in selecting Python.

The parser and lexer compiler components can be generated from the formal language specification, thus, a generator with Python target language support was needed to be selected. Another important criteria included versatility, separation of output code from grammar and comprehensive documentation. For this research, ANTLR [12] has been selected. A viable alternative to it is PLY [13].

Multimedia is the third base component of the implementation, cross-platform and open-source projects with Python bindings were preferred. We have selected GStreamer [14] because its core does not depend on the type of processed data and it can be extended by plugins, which enable adding new formats, protocols or hardware support.

The architecture of the implemented two-pass compiler includes frontend and backend (Fig. 1).

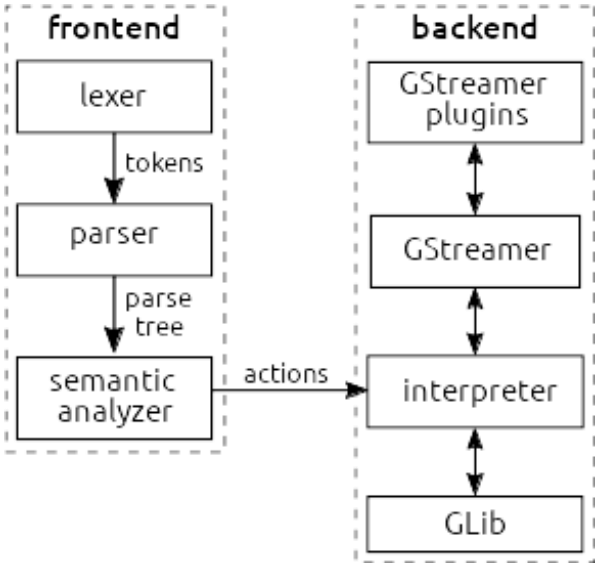


Fig. 1. Compiler architecture

2.1. Frontend. The frontend performs syntax and semantics verification (e.g., identifier availability, type checking). The lexer and parser are generated by ANTLR from the language context-free grammar description, it also generates the helper base classes for parse tree traversal: a listener and/or a visitor. Our semantic analyser implements the visitor design pattern to control the order of traversal.

All declarations for declarative sections are checked if a name they define has not been declared yet, and additional checks need to be performed for every section.

The parse tree traversal starts at the LIBRARIES section and the library importer component tries to import declared libraries as a regular Python package. The libraries which define custom types need to have *export_types* dict in their packages which maps type names to corresponding classes. If a library name has not been declared yet and its package was successfully imported, the library is registered in a corresponding lookup table and becomes available in anther program parts.

For declarations in the HANDLERS and RENDERERS sections, the visitor checks if the specified name has not declared yet, the library is imported, and it tries to import a specified element as a class in that library. If the class was imported successfully and inherits a required base class, it will be registered in the lookup table.

In SETS section, type import declarations are checked if a name has not been declared yet, the specified library was imported and has specified type, in that case the imported type is registered in the lookup table. Type alias are registered only for already present types if the name defined by an alias has not been used before.

The declarations in the ELEMENTS, TUPLES and AGGREGATES sections assign names to certain values in the lookup table when a specified type name or a referenced value name is defined.

The ACTIONS section declarations define program logic and data flows, they are converted to a hierarchy of action objects which represent specific discrete tasks.

The interface defined by the base action class includes methods *Start*, *Stop* and *Stop_Prepare*, *Finished* signal and *Depth* property (a depth within the hierarchy tree). The *Start* method returns False when the action finished immediately, otherwise it returns True. The *Stop_Prepare* method is used to notify an action before it is actually stopped with the *Stop* method call. The *Finished* signal is emitted only when the action finished neither during *Start* nor during *Stop* method calls.

Another important abstract base class is *CompositeAction*: a group of actions which can be treated as a single action and may define custom execution order. There are two concrete implementations: *ChinedAction* which executes child actions sequentially and *ParallelAction*, which executes child actions concurrently. When they are stopped, they stop child actions. The *ChainedAction* finishes execution only when all children have finished. The *ParallelAction* behaves in the same way when execution duration is not specified or when early finish is allowed, otherwise it continues execution even when all children have finished already and stops unfinished when the data processing time elapses.

To create a delay in the beginning of actions (e.g., in TIMELINE statements with non-zero start time), the *DelayAction* is used.

The behaviour which defines an update frequency and an elapsed time tracking is extracted into *ActionUpdateStrategy* subclasses and can be used in certain classes with help of the Dependency Injection design pattern. The base interface provides *update* method which receives a time difference for a current update step (for base class it updates an object age) and properties to get a current update interval duration, a next update delay, an overall duration, an age and an update loop finish status. All time values are stored with microsecond precision. There are two concrete implementations: *FixedStepUpdate* which has equal update intervals and *FramesTupleUpdate* which accepts a list of update intervals. The *DelayAction* objects instantiate and encapsulate *FixedStepUpdate* objects with an overall duration and an update interval duration equal to the specified delay. The *ParallelAction* can be initialized with any concrete implementation of *ActionUpdateStrategy* to define the action duration and the update frequency.

The *DownloadAction*, *UploadAction*, and *RenderAction* implement corresponding program statements DOWNLOAD FROM WITH, UPLOAD TO WITH, and RENDER WITH. They check compatibility of handlers and renderers with user-specified data stream objects, establish connections at the action start and disconnect them when stopped of finished execution. Additionally, *DownloadAction* and *UploadAction* create GStreamer pipeline element that handles input (or output) to a supplied source (or target) URL.

The action hierarchy is built by *TimelineBuilder* object and helper classes. Every TIMELINE statement block and root code block are treated as a separate time interval and are represented by separate *_Timeline* object within *TimelineBuilder*. These objects encapsulate checks for detecting duplicated data streams in lists, cases when the same data stream has different target or source declared for the same execution time, ensure compatibility between specified data streams, handler (or renderer) and action.

TimelineBuilder holds at least one *_Timeline* instance: for the root code block. Entering to a new TIMELINE statement block will create a new *_Timeline* instance with a parent object which becomes current while the previous one becomes its parent. Each *_Timeline* object has an initial mode set to "in parallel", but when the code block marked with SEQUENCE keyword is entered, it is switched to "in sequence". On exit from the top-level SEQUENCE block, all created actions within that block are put into wrapper *ChainedAction* (this step is skipped for single action) and are added to the actions list of this *_Timeline*. When TIMELINE statement block is exited, the current *_Timeline* instance is set to its parent. Empty TIMELINE blocks are ignored, but when time range is set and parent *_Timeline* has mode "in sequence", *DelayAction* is added instead to the parent. When time range is not specified for the current TIMELINE statement block and the parent has "in parallel" mode, all actions of the current *_Timeline* are pushed to the parent as they are. In all other cases they are put to *ParallelAction* which has an appropriate *ActionUpdateStrategy* set if needed. When non-zero start time is specified, *DelayAction* is created with the corresponding delay set. If parent has "in sequence" mode or start time is zero *ParallelAction* with *DelayAction* are pushed as is to the parent. Otherwise, they are wrapped with *ChainedAction* before they are added to the parent.

2.2. Backend. The backend provides runtime environment and runtime error handling, schedules execution of the action hierarchy received from the frontend stage. The action hierarchy is represented here by a single root *ParallelAction* object. The core component of the backend is the scheduler object *UpdateSource* which attach itself to GLib main context as an event source to receive updates.

Actions which need updates (e.g., *ParallelAction* and *DelayAction*) hold reference to the scheduler object and subscribe for updates when needed. The scheduler holds the priority queue of subscribed actions where priority determined by both the next update time in microseconds and the action depth within the hierarchy tree. This ensures that the parent action receives update before its children when they have the same update time. But for sibling actions update order (and, as a

result, the start order) is not guaranteed.

The scheduler update loop has three distinct stages (Fig. 2): *prepare*, *wait*, and *dispatch*.

The *prepare* stage determines the closest update time for subscribed actions and calculates a delay for the *wait* stage. On the first run, this stage starts a root action and the scheduler subscribes to the root action finished signal if it did not finish immediately, otherwise the *wait* stage is skipped.

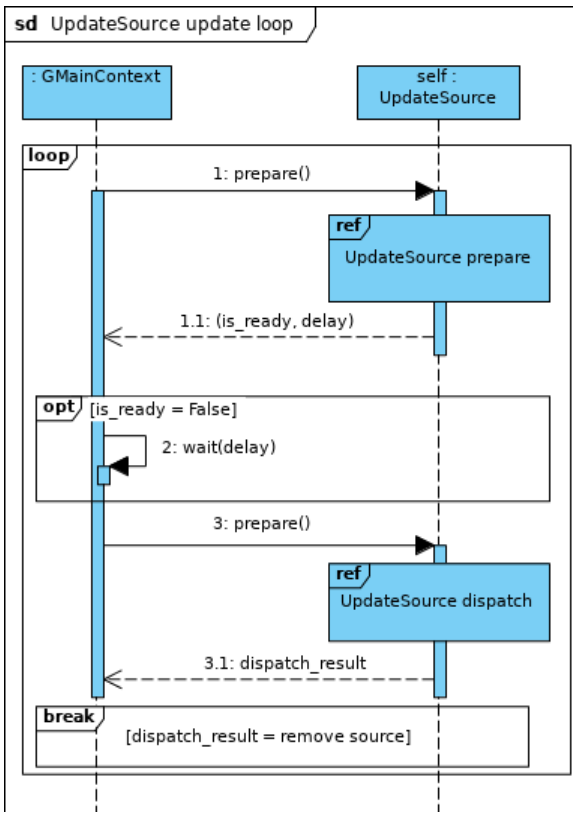


Fig. 2. Scheduler update loop

The *dispatch* stage pulls actions which reached their update time from a priority queue. The actions update method receives a defference between planned and actual update time and returns a Boolean value. An action which still needs updates is added back to the scheduler priority queue. At the end of this stage, *on_update_unsubscribed* handler is called for all unsubscribed actions in the order they received updates. If the root action is finished during or before the dispatch stage, the scheduler removes itself from GLib main context and the program finishes its execution.

2.3. Standard library. The standard library provides base classes and reference implementations of handlers, renderers and data streams. Handlers and renderers implement *BaseHandler* class, its interface define methods for attaching and detaching to data streams, compatibility checking, starting and stopping operation, signal *done* to notify subscribed objects. Renderers are always used as destination for data streams, while handlers can implement either source or destination. Depending on the implementation, they can be attached to single or multiple data streams at the same time (e.g., handlers for multiplexed data formats). The reference handler implementations are classes OGGIn and OGGOut which work with files in OGG container format and support the following data stream types: audio, video, and subtitles. Both handlers try to automatically find the required coder or decoder among GStreamer plugins when attached to uncompressed data streams.

Standard renderers are represented by *AudioOutput* and *VideoOutput* classes which implement playback of uncompressed audio and video. Data streams can have at most one source and multiple receivers (handlers and renderers). The base data stream class *BaseStream* provides an interface with the methods for attaching and detaching source and receiver objects, retrieving information about supported data types for its inputs and outputs, signals to notify when no outputs are present anymore or first output attached. The standard library provides implementation of uncompressed audio and video streams: *RawAudio* and *RawVideo*.

All these components have own GStreamer elements connected into pipelines. During runtime actions they are connected together to ensure data flow described by the program.

3. Results and discussion

In order to evaluate the efficiency of the implemented solution we compared it with traditional development tools, namely, C compiler from GNU Compiler Collection suite [15] and GStreamer framework. For this purpose, we created two sets of equivalent small- and medium-sized programs written in both C and ASAMPL. To evaluate performance of the implemented compiler, we

measured overall running time and peak memory consumption during program code parsing, analysis, and preparation program code to execution.

Table 1

Average program code size

Program code	Small size (lines of code)	Medium size (lines of code)
C	193	318
ASAMPL	43	95

For running time measurement, we used Perf tool [16], running each compiler for every program 1000 times. Averaged results for both program sets (see Table 2 and Fig. 3b) show that execution time of GCC C compiler is around 30 % lower.

Table 2

Average execution time

Program code	Small size (secs and stddev)	Medium size (secs and stddev)
GCC	0.220516528 (0.34%)	0.23745095 (0.34 %)
ASAMPL compiler	0.315360984 (0.53 %)	0.330313517 (0.29 %)
Difference	30 %	28 %

The results of peak memory consumption measurement with Valdgrind tool [17] (see Table 3 and Fig. 3c) show that on average it is 33% lower for ASAMPL compiler.

ASAMPL programs are visually simpler, concise, and expressive since they omit resource management, pipeline manipulation, synchronization and other tasks required to set up an environment, but not related to actual data processing task. As a result, the program size in lines-of-code (excluding comments and whitespace) for ASAMPL is at least 3 times smaller (see Table 1 and Fig. 3a).

Table 3

Peak memory consumption

Program code	Small size (Mbytes)	Medium size (Mbytes)
GCC	14.7	14.7
ASAMPL compiler	9.62	9.8
Difference	35 %	33 %

.

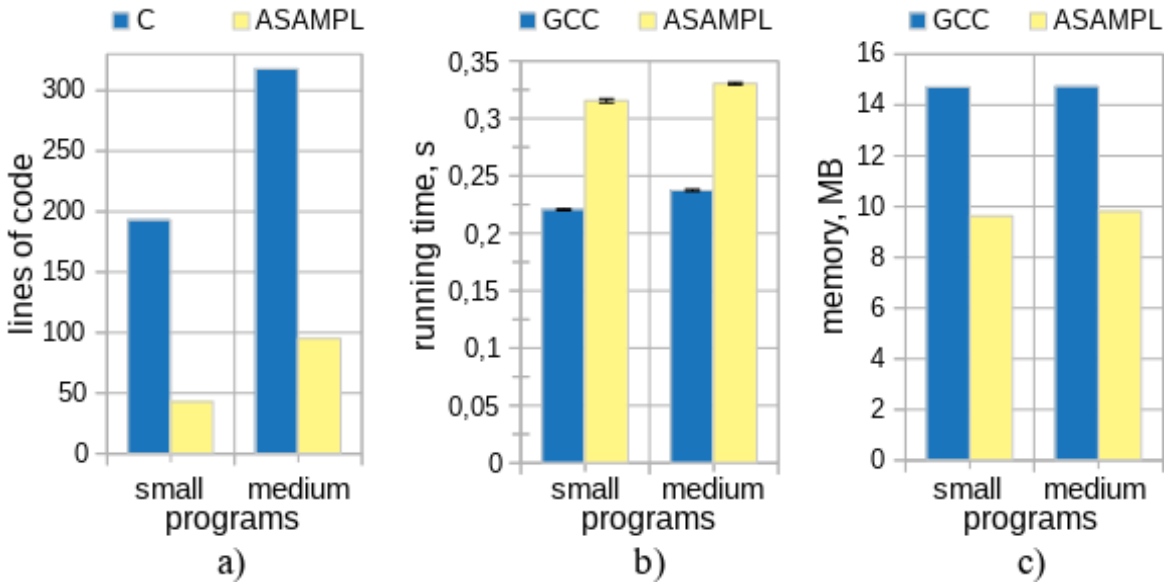


Fig. 3. Benchmarking results comparison charts

Conclusion

The programming language ASAMPL is a domain-specific language driven by data modality and time. It is developed for easy and effective processing of multimodal data defined with respect to time. To present such data, the programming language specification includes data structures called tuples and aggregates. ASAMPL offers a set of special-purpose operators, semantics of which is presented in the paper. To execute program code in ASAMPL, a complier has been developed. The paper presents both the architecture of this compiler and the compilation approach. The testing results show that ASAMPL code is simpler, concise, and expressive as well as memory consumption is lower for the developed ASAMPL compiler. Thus, ASAMPL can be considered as an appropriate option when the memory usage and the code simplicity are important issues.

References

1. Virtual Reality for Education. Available from: http://virtualrealityforeducation.com/ [Accessed 05/02/2020].
2. Antonov S., Antonova R., Spassov K. Multimedia Applications in Education. *Smart Technologies and Innovation for a Sustainable Future*. Springer. 2019. P. 263–271.
3. Weng C., Rathinasabapathi A., Weng A., Zagita C. Mixed Reality in Science Education as a Learning Support: A Revitalized Science Book. *Journal of Educational Computing Research*. 2018. 57(3). P. 777–807.
4. Sulema Y. ASAMPL: Programming Language for Mulsemedia Data Processing Based on Algebraic System of Aggregates. *Interactive Mobile Communication Technologies and Learning*. Springer. 2018. P. 431–442.
5. Dychka I., Sulema Ye. Logical Operations in Algebraic System of Aggregates for Multimodal Data Representation and Processing. *KPI Science News*. 2018. Vol. 6. P. 44–52.
6. Dychka I., Sulema Ye. Ordering Operations in Algebraic System of Aggregates for Multi-Image Data Processing. *KPI Science News*. 2019. Vol. 1. P. 15–23.
7. Sulema Ye., Kerre E. Multimodal Data Representation and Processing Based on Algebraic System of Aggregates, preprint. 2020. 37 p.
8. Milner R. Operational and Algebraic Semantics of Concurrent Processes. *Formal Models and Semantics*. 1990. P. 1203–1242.
9. Roşu G., Ştefănescu A. Towards a Unified Theory of Operational and Axiomatic Semantics. *Automata, Languages, and Programming*. Springer. 2012. P. 351–363.
10. TIOBE The Software Quality Company. Available from: https://www.tiobe.com/tiobe-index/ [Accessed 05/02/2020].
11. The Top Programming Languages 2019. IEEE Spectrum. Available from: https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019 [Accessed 05/02/2020].
12. ANTLR. Available from: https://www.antlr.org/ [Accessed 05/02/2020].
13. PLY (Python Lex-Yacc). Available from: https://www.dabeaz.com/ply/[Accessed 05/02/2020].
14. GStreamer. Available from: https://gstreamer.freedesktop.org/ [Accessed 05/02/2020].
15. GCC, the GNU Compiler Collection. Available from: https://gcc.gnu.org/ [Accessed 05/02/2020].
16. Perf Wiki. Available from: https://perf.wiki.kernel.org/index.php/Tutorial [Accessed 05/02/2020].
17. Valgrind's Tool Suite. Available from: https://valgrind.org/info/tools.html [Accessed 05/02/2020].

Література

1. Virtual Reality for Education. Available from: http://virtualrealityforeducation.com/ [Accessed 05/02/2020].
2. Antonov S., Antonova R., Spassov K. Multimedia Applications in Education. Smart Technologies and Innovation for a Sustainable Future. Springer. 2019. P. 263–271.
3. Weng C., Rathinasabapathi A., Weng A., Zagita C. Mixed Reality in Science Education as a Learning Support: A Revitalized Science Book. Journal of Educational Computing Research. 2018. 57(3). P. 777–807.
4. Sulema Y. ASAMPL: Programming Language for Mulsemedia Data Processing Based on Algebraic System of Aggregates. Interactive Mobile Communication Technologies and Learning. Springer. 2018. P. 431–442.
5. Dychka I., Sulema Ye. Logical Operations in Algebraic System of Aggregates for Multi-

modal Data Representation and Processing. KPI Science News. 2018. Vol. 6. P. 44–52.
6. Dychka I., Sulema Ye. Ordering Operations in Algebraic System of Aggregates for Multi-Image Data Processing. KPI Science News. 2019. Vol. 1. P. 15–23.
7. Sulema Ye., Kerre E. Multimodal Data Representation and Processing Based on Algebraic System of Aggregates, preprint. 2020. 37 p.
8. Milner R. Operational and Algebraic Semantics of Concurrent Processes. Formal Models and Semantics. 1990. P. 1203–1242.
9. Roşu G., Ştefănescu A. Towards a Unified Theory of Operational and Axiomatic Semantics. Automata, Languages, and Programming. Springer. 2012. P. 351–363.
10. TIOBE The Software Quality Company. Available from: https://www.tiobe.com/tiobe-index/ [Accessed 05/02/2020].
11. The Top Programming Languages 2019. IEEE Spectrum. Available from: https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019 [Accessed 05/02/2020].
12. ANTLR. Available from: https://www.antlr.org/ [Accessed 05/02/2020].
13. PLY (Python Lex-Yacc). Available from: https://www.dabeaz.com/ply/[Accessed 05/02/2020].
14. GStreamer. Available from: https://gstreamer.freedesktop.org/ [Accessed 05/02/2020].
15. GCC, the GNU Compiler Collection. Available from: https://gcc.gnu.org/ [Accessed 05/02/2020].
16. Perf Wiki. Available from: https://perf.wiki.kernel.org/index.php/Tutorial [Accessed 05/02/2020].
17. Valgrind's Tool Suite. Available from: https://valgrind.org/info/tools.html [Accessed 05/02/2020].

Received 06.02.2020

About the authors:

Yevgeniya Sulema,
candidate of tech. sciences (PhD),
associate professor at Computer Systems Software Department.
The number of publications in Ukrainian and foreign journals is over 130.
Hirsh index is 4.
https://orcid.org/0000-0001-7871-9806,

Vladyslav Glinskii,
Master student at Computer Systems Software Department.
https://orcid.org/0000-0003-1360-7218.

Affiliation:

Igor Sikorsky Kyiv Polytechnic Institute,
03056, Kyiv,
pr. Peremogy, 37, build. 15.
Tel.: +38 044 204 99 44.

Email: sulema@pzks.fpm.kpi.ua

УДК 004.424

Опис списків і множин в метамові нормальних форм знань / О.П. Кургаєв. – С. 3 – 16.

Списки використовують для подання всіляких знань. У вигляді списків зручно представляти формули, функції, дерева, графи, множини й багато інших складних об'єктів. Множина – одна з найбільш важливих структур даних, використовуваних як у математиці, так і в програмуванні. Запропоновано формалізацію у метамові нормальних форм знань списків, предикатів на списках і множинах, базуючись на відомих Пролог-формалізаціях цих понять, що використовують списковий домен. Серед предикатів на списках описано: додавання елемента, видалення елемента, пошук останнього елемента, пошук сусідніх елементів, конкатенація списків, реверс, паліндром, видалення всіх входжень елемента і ін. Використовуючи списковий домен описано предикати на множинах: перетворення списку в множину, приналежність елемента множині, об'єднання, перетин, різниця, симетрична різниця, збіг, доповнення множин, відношення підмножини, власної підмножини.

Ключові слова: метамова, список, множина, предикат, рекурсія, визначення.

UDC 004.424

The description of lists and sets in the meta-language of normal forms of knowledge / A.F. Kurgaev. – P. 3 – 16.

Lists use for representation of various knowledge. As lists it comfortably to present formulas, functions, trees, columns, great numbers and many other difficult objects. Great number - one of the most essential structures of data, used both in mathematics, and in programming. The formalization of lists, list-based predicates and set-based predicates in the meta-language of normal forms of knowledge is presented, based on the known Prolog-formalizations of these concepts, which use a list-domain. Among the described list-based predicates are the following: adding an element to the list, removing an element, finding the last element of a list, finding adjacent elements in a list, concatenation of lists, reversing a list, palindrome, etc. Using the list-domain, the set-based predicates are described as follows: converting a list into a set, checking if an element is in a set, concatenation, intersection, difference, symmetrical difference, identity, complement of sets, relation of subset, proper subset.

Key words: meta-language, set, list, predicate, recursion, definition.

УДК 004.4'24

Автоматизированное проектирование программ для платформы .NET, использующих библиотеку параллельных задач / А.Е. Дорошенко, Е.А. Яценко. – С. 17 – 24.

Необходимость в повышении производительности программного обеспечения для решения трудоемких задач, с одной стороны, и новые возможно-

UDC 004.4'24

Automated design of programs for .NET platform using Task Parallel Library / A.Yu. Doroshenko, O.A. Yatsenko. – P. 17 – 24.

The necessity to improve the performance of software solving labour-intensive tasks, on the one hand, and new capabilities provided by multicore archi-

сти распараллеливания вычислений, предоставляемые многоядерной архитектурой современных микропроцессоров, с другой, побуждает к созданию специализированных инструментальных средств для разработки параллельных программ для таких архитектур. Одним из способов дальнейшего повышения эффективности многопоточных программ, разработанных на платформе .NET, является использование библиотеки параллельных задач TPL. В статье выполнено дальнейшее развитие ранее разработанного алгебро-алгоритмического инструментария в направлении формализованного проектирования и синтеза параллельных программ на языке C#, использующих средства TPL. Упомянутая библиотека повышает производительность труда разработчиков за счет упрощения процедуры добавления параллелизма в программу и динамически масштабирует степень параллелизма для наиболее эффективного использования всех доступных процессоров. В основу предлагаемого подхода положены язык систем алгоритмических алгебр Глушкова и метод конструирования синтаксически правильных программ, который исключает возможность появления синтаксических ошибок в процессе проектирования схем. Проведен эксперимент по выполнению сгенерированных с помощью разработанного инструментария примеров параллельных программ на многоядерном процессоре.

Ключевые слова: автоматизированное проектирование программ, алгебра алгоритмов, библиотека параллельных задач, многопоточность, параллельные вычисления, синтез программ, TPL.

tecture of contemporary microprocessors, on the other, encourages the development of specialized software tools for automated development of parallel programs for such architectures. Further progress in improving the efficiency of multithreaded programs on .NET platform is using the task parallel library TPL. The paper proposes the further development of previously developed algebra-algorithmic tools in the direction of formalized design and synthesis of C# programs using TPL. The library raises the labour productivity of developers by simplifying the procedure of adding parallelism to a program and dynamically scales parallelism level to use all available processors in the most efficient way. The proposed approach uses high-level language based on Glushkov's system of algorithmic algebra and the method of designing syntactically correct programs that excludes the possibility of appearance of syntactic errors during scheme design. The results of the experiment consisting in executing examples of generated parallel programs on a multicore processor are given.

Key words: automated software design, algorithm algebra, multithreading, parallel computation, software synthesis, task parallel library, TPL.

УДК 004.82

UDC 004.82

Про один метод витягу даних з слабоструктурованих документів / К.О. Кудім, Г.Ю. Проскудіна. – С. 25 – 32.

A method for extracting data from semistructured documents / K.A. Kudim, G.Yu. Proskudina. – P. 25 – 32.

В роботі розроблений, докладно описаний і практично випробуваний лінгвістичний метод вирішення задачі ви-

Linguistic method to solve the problem of data extraction from weakly structured documents is developed, approved, and

тягу даних на прикладі витягу даних про персоналії з слабоструктурованих документів, представлених в загальнодоступному каталозі авторефератів дисертацій Національної бібліотеки України ім. В.І. Вернадського. Описана вся послідовність кроків: вибір колекції документів; підготовка документів; написання правил граматики для отримання даних з тексту; написання правил перевірки морфології; створення інтерпретацій або прив'язок правил до даних; аналіз результатів розбору. Лінгвістичний метод витягу даних виявив ряд переваг в порівнянні з описаним раніше методом отримання даних за допомогою регулярних виразів.

Ключові слова: слабоструктуровані документи, витяг інформації, лінгвістичний аналізатор, синтаксичний аналізатор, морфологічний аналіз, контекстно-вільна граматика.

УДК 004.853, 004.55

Разработка методов управления доступом к информации в Wiki-ресурсах / И.Ю. Гришанова, Ю.В. Рогушина. – С. 33 – 46.

Рассмотрены перспективы и область применения Wiki-технологий, проанализирована структура представления контента и программное обеспечение MediaWiki. Значительное внимание отводится архитектуре MediaWiki и компонентам этой архитектуры. Определена проблема, связанная с необходимостью управления доступом к контенту Wiki-ресурсов в соответствии со спецификой информации, которая содержится в таких ресурсах. Результаты анализа показали, что базовые средства MediaWiki не позволяют получить удовлетворительное решение этой задачи. Поэтому возникает потребность в создании специализированного программного обеспечения, которое базируется на классификации контента с использованием отдельных пространств имен, категорий, шаблонов и семантических свойств. которые извлекаются из MediaWiki, и являются независимыми от ядра MediaWiki, а базируется на

described in detail in the paper. Sample data were taken from thesis catalogue of Vernadsky National Library of Ukraine. The sequence of all stages is described: document collection choice; document preparation; writing grammar rules for data extraction from text; writing rules for morphology verification; creation of interpretations or bindings rules to data; analysis of parsing results. Linguistic method of data extraction showed many advantages in comparison to the method of data extraction with regular expressions described earlier.

Key words: weakly structured documents, information extraction, linguistic analyzer, syntactic analyzer, morphological analysis, context-free grammar.

UDC 004.853, 004.55

Development of access management methods to information from Wiki resources / I.Y. Grishanova, J.V. Rogushina. – P. 33 – 46.

The prospects and scope of Wiki technologies are considered, the structure of the content re presentation and MediaWiki software are analyzed. Considerable attention is given to architecture of MediaWiki and the components of this architecture. The problem deal with the need to access management to the content of Wiki-resources in accordance with the specifics of the information contained in such resources is identified. The results of the analysis show that the basic tools of MediaWiki are not capable to allow a satisfactory solution to this problem. Therefore, we need to create specialized software that is based on the content classification with the use of separate namespaces, categories, templates and semantic properties extracted from MediaWiki. Such software has to be independent of the MediaWiki core and be based on programming of content analysis at the skin level. We test the proposed solution in the process of development of the portal version of the

программировании анализа контента на уровне скина. Приведенное решение апробировано в разработке портальной версии Большой украинской энциклопедии (е-вуе) и использует знания относительно структуры базы знаний этого портала.

Ключевые слова: MediaWiki, Wiki-ресурс, управление доступом, Большая украинская энциклопедия.

Great Ukrainian Encyclopedia (e-vue) with use of knowledge regarding the knowledge base structure of this portal.

Key words: MediaWiki, Wiki resource, access management, Great Ukrainian Encyclopedia.

УДК 004.05, 004.4[2+9], 004.94, 519.7

UDC 004.05, 004.4[2+9], 004.94, 519.7

Алгебраические шаблоны уязвимостей двоичного кода / В.М. Яковлев. – С. 47 – 54.

Algebraic patterns of binary code vulnerabilities / V.M. Yakovlev. – P. 47 – 54.

Одной из актуальных проблем в ИТ-индустрии является обнаружение уязвимостей в программном обеспечении. В последнее десятилетие стали весьма популярными и многообещающими подходы к решению данной проблемы, основанные на символьных методах. Данная статья описывает подход к поиску уязвимостей в двоичном коде, основанный на формальных методах символьного моделирования и алгебраического сопоставления. В данной статье предлагается формализация представления двоичного кода и уязвимостей на основе алгебры поведений, а также методика формирования шаблонов уязвимостей. Такие шаблоны могут быть использованы не только для описания, но и для поиска уязвимостей в произвольном двоичном коде. Алгебра поведений используется для представления поведения как двоичного кода, так и уязвимости. Однако, хотя получение представления бинарного кода в терминах алгебры поведений может быть автоматизировано, создание описания уязвимостей требует разработки корректной и эффективной методологии. При использовании представлений в терминах алгебры поведений задача поиска уязвимостей может быть решена в два этапа – относительно быстрого алгебраического сопоставления и собственно символьного моделирования на основе данных, полученных на этапе сопоставления. Разработка описаний уязвимостей в тер-

Detection of software systems' vulnerabilities is an actual problem in the IT industry nowadays. The approaches to the solution of this problem, based on the symbolic methods, became very popular and promising during the last decade. The article describes an approach to the vulnerabilities detection in the binary code, based on the formal methods of symbolic modeling and algebraic matching. In the article, the formalization of representation of binary code and vulnerabilities based on the behavior algebra, and the method of creation of formal patterns of vulnerabilities are proposed. The behavior algebra used for the representation of the formal binary code behavior, as well as for describing the vulnerabilities behavior. However, while the representation of the binary code in the terms of behavior algebra could be automated, creation of the vulnerabilities description requires development of the correct and effective methodology. Using the behavior algebra representation, the task of vulnerabilities detection can be solved in two steps – relatively fast algebraic matching, and the symbolic modeling itself, based on the data provided by the algebraic matcher. By the development of the vulnerabilities description in the terms of behavior algebra, and the algebraic matching algorithm the speed of detection of vulnerabilities in the binary code can be increased. The methodology of development of the vulnerabilities description in the terms of the behavior algebra has been proposed. The advantage

минах алгебры поведений и алгоритма алгебраического сопоставления позволяет ускорить алгоритмы поиска уязвимостей в двоичном программном коде. Предложена методика разработки описаний уязвимостей двоичного кода в терминах алгебры поведений. Преимущество алгебраического подхода состоит в том что уязвимости в коде могут быть определены более точно, а описание уязвимости в терминах алгебры поведений позволяет учесть ее различные варианты. Кроме того, эксперименты с прототипом показали, что «двухуровневая» система поиска уязвимостей работает быстрее, чем «чистая» система символьного моделирования: вначале выполняется быстрый этап сопоставления, а затем медленный этап символьного моделирования на данных, полученных на предыдущем этапе.

Ключевые слова*:* уязвимости программного обеспечения, символьное моделирование, алгебраическое сопоставление, алгебра поведений.

of the algebraic approach is that the code vulnerabilities can be found more precisely, and the vulnerability description in the terms of behavior algebra can take in account different possible kinds of it. Also, the experiments with the implementation prototype shown that the "two-level" vulnerability detection system is faster than "pure" symbolic modeling: the fast matching step is executed first, and the slow modeling step is executed next on the results, provided by the matching step.

Key words: software vulnerabilities, symbolic modeling, algebraic matching, behavior algebra.

УДК 004.05

Децентрализированные системы в логистике: обзор использования и проблемы безопасности / А. Летичевский**,** С. Горбатюк. – С. 55 – 73.

Проанализировано направления применения и использования децентрализованных (распределенных) систем на примере технологии блокчейн в международной логистике. Рассмотрены перспективы и ключевые аспекты дальнейшего развития децентрализованных систем и больших баз данных. Рассмотрены основы функционирования смарт-контрактов на базе Etherium и языка Solidity. Проанализированы основные проблемы безопасности в целостной цепи поставки, выявлены направления улучшения отслеживания непрерывности контроля качества товара при движении в мультимодальной цепи поставок. Предложены методы усиления контроля за безопасностью в децентрализованных системах в логистике. Рассмотрена методология формальных алгебраических методов для анализа и исследова-

UDC 004.05

Decentralized Systems in Logistics: Usage Overview and Security Issues / O. Letychevskyi, S. Horbatiuk. – P. 55 – 73.

The directions of the application and use of decentralized (distributed) systems are analyzed using the example of blockchain technology in the international logistics. The prospects and key aspects of the further development of decentralized systems and large databases are considered. The basics of the functioning of smart contracts based on Etherium and Solidity language are considered. The main security problems in the integrated supply chain are analyzed, and directions for improving the monitoring of the continuity of product quality control during movement in a multimodal supply chain are identified. Methods are proposed for strengthening security control in decentralized systems in logistics. The methodology of formal algebraic methods is considered in order to analyze and study the properties of transportations in the interac-

ния свойств перевозок при взаимодействии агентов определенной логистической среды.

Ключевые слова: децентрализованная система, блокчейн, смарт-контракт, логистика, Solidity.

tion of agents of a certain logistic environment.

Key words: decentralized system, blockchain, smart contract, logistics, Solidity.

УДК 004.432

Семантика та прагматика мови програмування ASAMPL / Є.С. Сулема, В.В. Глінський. – С. 74 – 83.

У цій статті представлено семантику проблемно-орієнтованої мови програмування ASAMPL та практичну реалізацію компілятора для неї. Ця мова програмування була розроблена для забезпечення ефективного оброблення мультимодальних даних, зокрема, оброблення мультимедійного контенту, компоненти якого явно визначені на часовій шкалі. Концепція оброблення даних, яка використовується в ASAMPL, заснована на структурах даних, операціях та відношеннях, визначених у алгебраїчній системі агрегатів. У статті представлено основні семантичні конструкції мови, які використовуються для оброблення даних. Крім того, в статті пояснюється підхід до компіляції програм на мові програмування ASAMPL, а також представлено результати тестів. Для порівняння результатів, отриманих для мови програмування ASAMPL, тестування проводилося також для аналогічних програм, написаних на мові програмування C, компіляція яких виконувалась за допомогою компілятора GCC. Важливим результатом тестування є підтвердження гіпотези про те, що мова програмування ASAMPL дозволяє розробляти більш компактний та зрозумілий програмний код, виконання якого вимагає менше пам'яті.

Ключові слова: мультимедіа, мультимодальні дані, мова програмування, компіляція.

УДК 004.432

Семантика и прагматика языка программирования ASAMPL / Е.С. Сулема, В.В. Глинский. – С. 74 – 83.

В данной статье представлены семантика проблемно-ориентированного языка программирования ASAMPL и практическая реализация его компилятора. Этот язык программирования был разработан для обеспечения эффективной обработки мультимодальных данных, в частности, обработки мультимедийного контента, компоненты которого явно определены на временной шкале. Концепция обработки данных, используемая в ASAMPL, основана на структурах данных, операциях и отношениях, определенных в алгебраической системе агрегатов. В статье представлены основные семантические конструкции языка, которые используются для обработки данных. Кроме того, в статье объясняется подход к компиляции программ на языке программирования ASAMPL, а также представлены результаты тестов. Для сравнения результатов, полученных для языка программирования ASAMPL, тестирование проводилось также для аналогичных программ, написанных на языке программирования C, компиляция которых производилась с помощью компилятора GCC. Важным результатом тестирования является подтверждение гипотезы, что язык программирования ASAMPL позволяет разрабатывать более компактный и понятный программный код, исполнение которого требует меньше памяти.

Ключевые слова: мультимедиа, мультимодальные данные, язык программирования, компиляция.

ДО УВАГИ АВТОРІВ!

У журналі "Проблеми програмування" публікуються наукові матеріали, які раніше не публікувалися в інших виданнях.

Мова статті: українська, російська, англійська. Обсяг статті - від 6 до16 сторінок формату А4.

Документ зберігається у форматі doc або docx. Ім'я подається транслітерацією, як прізвище автора (авторів), наприклад, “Petrenko.doc”.

Автори можуть користуватися електронною поштою і також телефаксом для ділової переписки та передачі до редакції тексту статті та правки при коректурі. E-mail редакції: alengoro@isofts.kiev.ua. FAX: +380 (44) 526 6263, Телефон: 526 5065.

1. Оформлення файлу з текстом статті.

При підготовці файлу використовуються: стиль нормальний (звичайний) або normal; шрифт Times New Roman, розмір шрифта 12 пт.; міжрядковий інтервал – 1,0; абзацний відступ – 1,25 см; вирівнювання – по ширині. У тексті не допускається вирівнювання пропусками; розстановка переносів – автоматична. Формат паперу А4, розміри полів документа – 20 мм. Текст статті після анотації має бути оформлений **у 2 колонки,** ширина яких – 7,86 см, а пробіл між ними – 1,27 см.

2. Послідовність розміщення та оформлення матеріалу статті.

УДК: індекс за універсальною десятковою класифікацією.

Автори: ініціали та прізвища авторів, курсив (світлий).

Заголовок 1 **(назва статті):** не містить абревіатур та строго відповідає змісту статті. Шрифт 15 пт, напівжирний, регістр верхній.

Анотація **(мовою статті):** 50-100 слів, не містить абревіатур, зрозумілих із змісту статті. Шрифт 10 пт, звичайний.

Ключові слова **(мовою статті):** не більше10 слів, не містить абревіатур, зрозумілих із змісту статті, подаються в називному відмінку, розділені комами. Шрифт 10 пт, звичайний.

Заголовок 2 **(назва розділу):** шрифт 14 пт, напівжирний; абзац із центральним вирівнюванням, без переносів. Заголовки нижчого рівня (пункти і т.п.) у самостійний абзац не виділяються і проходять першим реченням текстового абзацу, шрифт 12 пт, напівжирний.

Основний текст статті має такі необхідні елементи:

постановка проблеми в загальному вигляді і її зв'язок з важливими науковими або практичними завданнями;

аналіз останніх досліджень і публікацій, у яких розпочато рішення даної проблеми і на які спирається автор, виділення невирішених раніше частин загальної проблеми, яким присвячується дана стаття;

формулювання цілей статті (постановка задачі);

виклад основного матеріалу дослідження з повним обґрунтуванням отриманих наукових результатів;

висновки з даного дослідження і перспективи подальших розробок у даному напрямку;

подяка (за наявності такої).

Формули створюються в редакторі Microsoft Equation 3.0 або MathType. Формули, на які є посилання в тексті, повинні мати наскрізну нумерацію. Номер формули друкується в круглих дужках біля краю правого поля. Розмір основного шрифту редактора формул – 12 пт. Розміри символів у формулах: звичайний – 12 пт, великий індекс – 9 пт, дрібний індекс – 7 пт, великий символ – 18 пт, дрібний символ – 11пт. Не допускається масштабування формульних об´єктів.

Рисунки мають бути створені вбудованим редактором Word Picture або експортовані з прикладних програм Windows у графічних форматах (bmp, pcx, gif, jpg або tif). Рисунки розташовуються по центру. Нумерація рисунків здійснюється відповідно до порядку

згадування у тексті. Нумеровані підписи розміщуються під рисунком з позначенням «Рис. », далі вказується номер рисунка і текст підпису.

Таблиці мають бути підготовлені стандартним вбудованим в Word інструментарієм "Таблиця". Таблиці нумеруються за порядком згадування. На номер таблиці повинно бути посилання в тексті. Номер таблиці вказується в окремому рядку з вирівнюванням по правій стороні (наприклад, «Таблиця 1»). Назви таблиць розміщуються над таблицею з вирівнюванням по центру. Мінімальний розмір шрифту в таблицях – 11 пт.

Література**:** нумерований список джерел згідно ДСТУ 8302:2015 від 01.07.2016 р., шрифт 11 пт, відступ: спеціальний, навислий, 0,63 см.

Література англійською мовою (References): список використовуваних джерел згідно ***Harvard Style***. Джерела з заголовками на латиниці наводяться без перекладу. Для літератури джерел на мовах, що не використовують латинський алфавіт, необхідно забезпечити переведення назв джерел і вказати після них у дужках мову оригіналу. Прізвища та ініціали авторів слід транслітерувати за правилами, як для закордонного паспорта. Приклади оформлення бібліографічних посилань згідно з вимогами ***Harvard Style*** наведені в багатьох публікаціях, наприклад, за електронною адресою http://www.staffs.ac.uk/assets/harvard_referencing_examples_tcm44-39847.pdf

Дані про авторів**:** мають починатися рядком "Про авторів:", напівжирний курсив. Далі вказуються для кожного з авторів ПІБ повністю, наукове звання, посада, адреса, кількість публікацій в українських виданнях (приблизна), кількість публікацій в зарубіжних індексованих виданнях (приблизна), індекс Хірша (за наявності), обов'язково номер ORCID (сайт ORCID http://orcid.org/).

Дані про місце роботи авторів**:** починаються рядком "Місце роботи авторів:", напівжирний курсив. Далі вказуються місце роботи, адреса, телефон, факс, електронна пошта, контактний телефон.

3. Оформлення файлу з анотаціями.

Файл з анотаціями містить інформацію двома мовами (наприклад, якщо стаття написана українською мовою, то анотації та ключові слова – російською та англійською мовами) та має бути оформлений у дві колонки: УДК (шрифт – 8 пт); назва статті (шрифт – 12 пт, напівжирний); прізвища та ініціали авторів (шрифт – 12 пт); текст анотації, ключові слова (шрифт – 10 пт).

Вимоги до анотації англійською мовою: обсяг від 100 до 250 слів, інформативність, оригінальність (не є калькою української або російськомовної анотації), змістовність (відображає основний зміст статті і результати досліджень), структурованість (дотримується логіки опису результатів у статті).

Документ зберігається у форматі doc або docx. Ім´я подається транслітерацією, як прізвище автора (авторів), наприклад, "Petrenko_Annot.doc".

Примітка: Підписний індекс журналу "Проблеми програмування" – **90853.**